Titre: Title:	Vérification fonctionnelle et validation de performance architecturale pour des tissus d'interconnexion
Auteur: Author:	Dany Lebel
Date:	2009
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Lebel, D. (2009). Vérification fonctionnelle et validation de performance architecturale pour des tissus d'interconnexion [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8475/

UNIVERSITÉ DE MONTRÉAL

VÉRIFICATION FONCTIONNELLE ET VALIDATION DE PERFORMANCE ARCHITECTURALE POUR DES TISSUS D'INTERCONNEXION

DANY LEBEL
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
Août 2009

Your file Votre référence
ISBN: 978-0-494-57256-6
Our file Notre référence
ISBN: 978-0-494-57256-6

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

VÉRIFICATION FONCTIONNELLE ET VALIDATION DE PERFORMANCE ARCHITECTURALE POUR DES TISSUS D'INTERCONNEXION

présenté par: LEBEL Dany

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. DAVID Jean-Pierre, Ph.D., président

M. SAVARIA Yvon, Ph.D., membre et directeur de recherche

Mme. NICOLESCU Gabriela, Doct., membre et codirectrice de recherche

M. LANGLOIS Pierre, Ph.D., membre

DÉDICACE

À mes parents

REMERCIEMENTS

Plusieurs personnes ont contribué à ce que cette maîtrise soit un succès et que l'expérience acquise tant au niveau personnel que technique soit des plus enrichissante.

Pour commencer, merci à Yvon Savaria de m'avoir confié ce projet des plus intéressant et pertinent dans mon cheminement de carrière. Il a su m'épauler tout au long de la progression de mes travaux. Les qualités d'interlocuteur technique et de présentateur exceptionnel de ce dernier ne peuvent que me servir d'exemple.

Merci à Gabriela Nicolescu d'avoir permis à ce projet de se concrétiser. Son expertise logicielle et réseautique a contribué à son succès.

Merci également à Normand Bélanger pour son implication importante dans le projet. Ses conseils démontrant pleinement sa grande expérience dans le domaine ont souvent servi d'inspiration à l'évolution du projet. Sa grande disponibilité et les discussions reliées ou non au domaine furent très appréciées.

Un merci tout particulier à Maria Mbaye et Nicolas Beucher, mes voisins de bureau les plus présents. Maria a été d'une grande aide lors de mes débuts à collaborer avec Monsieur Savaria. Ses conseils, son savoir faire ainsi que les quelques heures passées ensemble sur les laboratoires de VLSI furent appréciés. Quant à Nicolas, sans contredit le plus fidèle au poste dans le bureau, ses commentaires d'un œil externe furent toujours pertinents. La réalisation d'un système qui intègre nos deux projets fut une belle expérience d'intégration. Il ne faut pas passer sous silence les nombreuses parties de badminton en compagnie de Nicolas et Mohamed qui nous ont permis de s'aérer l'esprit. Merci à Mohamed Rouatbi pour ses suggestions occasionnelles ainsi que pour les séances de « gym extrême » et sorties forts agréables.

Merci à Rahul, Gilbert, Vincent et tous les autres que j'ai côtoyés, d'avoir rendu cette expérience encore plus agréable.

Merci à mes parents de m'avoir encouragé et soutenu durant toutes mes études. Un merci fort mérité à Henriette pour son soutien quotidien et son amour.

RÉSUMÉ

L'exploration architecturale, la validation et la vérification sont parmi les défis les plus ardus en conception de circuits intégrés. Le développement d'un environnement adéquat pour toutes les phases d'un projet reste un défi de taille malgré l'avènement d'outils, langages et bibliothèques tel que le SystemC. Un tel environnement, qui s'intègre bien à la méthode de conception, est présenté dans ce mémoire. L'environnement inclut deux niveaux d'abstraction dans le but d'intégrer l'exploration architecturale et la vérification d'un design. Afin d'être flexible et complet, l'environnement inclut trois types de génération de données : pseudo-aléatoire contraint, synthétique et réelle, avec une application exécutée sur un processeur. Il est montré que la productivité est potentiellement accrue pour l'exploration architecturale et les phases ultérieures du développement. Ce projet traite de la validation de tissus d'interconnexion et implante, pour des fins de démonstration, deux types d'ordonnanceurs : « Weighted Round-Robin » et « Wrapped Wave Front Arbiter ». L'environnement est aisément modifiable pour d'autres contextes et besoins. Le but premier est de démontrer la pertinence de l'environnement et non de prouver l'efficacité d'une architecture en particulier. L'évaluation de l'équité sous un trafic non-uniforme pour les deux ordonnanceurs est, entre autres, utilisée comme exemple d'évaluation de performance.

ABSTRACT

Architectural exploration, validation, and design verification are currently among the toughest challenges in ASIC design. Devising an ASIC development environment to meet those needs remains a challenge even with current tools, languages, and libraries such as SystemC. In this document, such an environment, which integrates well to the design methodology, is presented. This environment supports two abstraction levels in order to allow smooth transition from architectural exploration and validation to design verification. In order to enable complete and flexible architecture validation, it also includes three types of data generation: constrained pseudo-random, synthetic, and traffic produced by a real application running on a processor. It is shown that the productivity is potentially increased when performing architectural exploration. The created environment focuses on validating switch fabric architectures and, in order to demonstrate its usage, two types of schedulers are implemented: Weighted Round-Robin and Wrapped Wave Front Arbiter. The environment can easily be modified for other contexts and needs. The primary goal is to show the usefulness of the ideas behind the environment instead of showing the performance of a specific architecture. Fairness evaluation under non-uniform traffic for two types of scheduler is used as one of the examples of performance evaluation.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ASIC	"Application-Specific Integrated Circuit », circuit intégré dédié à une application spécifique
ASIP	"Application-Specific Instruction-set Processor », processeur à jeu d'instructions spécialisé
AVM	"Advanced Verification Methodology", une méthode de vérification proposée par Mentor Graphics.
CI	Circuit intégré
DUV	"Design Under Verification", module sous vérification
FIFO	"First In, First Out", une organisation de mémoire sous forme de file qui préserve l'ordre d'entrée.
FS	File de Sortie.
FSM	"Finite State Machine", machine à états
HDL	"Hardware Description Language", langage de description matériel
MC-FRC	"Motion Compensated Frame Rate Conversion", augmentation du taux de trames par compensation de mouvement
OSCI	"Open SystemC Initiative", version gratuite de SystemC
PIM	"Platform Independent Model". Il s'agit de l'interface externe fournie par Tensilica pour communiquer avec leur processeur.
RTL	"Register Transfer Language", langage de transfert de registre tel que VHDL et Verilog
SCV	"SystemC Verification library", bibliothèque de vérification de SystemC

SoC	"System On Chip", Systèmes sur puce
TI	Tissu d'interconnexion
TLM	"Transaction Level Model", modèle transactionnel
TVM	"Transaction-Based Verification Model", modèle de vérification transactionnel
VIP	"Verification Intellectual Property", un module de vérification standard réutilisable.
UVM	"Unified Verification Methodology", Méthode de Vérification Unifiée de Cadence
VOQ	"Virtual Output Queues", Files de sortie virtuelles
WRR	"Weighted Round-Robin", un type d'arbitre
WWFA	"Wrapped Wave Front Arbiter", un type d'arbitre
XTSC	XTensa SystemC

LISTE DES ANNEXES

INTRODUCTION

Les tissus d'interconnexion (TI) sont largement utilisés pour les communications. Avec la complexité des circuits intégrés qui ne cesse d'augmenter, les TI augmentent en popularité dans les systèmes embarqués. Le protocole *rapid-io* est exploité dans une classe de TI populaire pour la réalisation des systèmes embarqués. Les TI peuvent, par exemple, être utilisés pour connecter plusieurs processeurs ou modules matériels dédiés avec des mémoires externes. Ce type d'architecture pourrait être amené à remplacer l'architecture basée sur un « bus », qui est bien connue mais qui possède un désavantage majeur : lorsque le bus est utilisé, le milieu de transmission est bloqué pour les autres communications qui n'ont possiblement aucun lien logique avec la communication active [37]. Les TI [28] ont l'avantage majeur de permettre plusieurs communications indépendantes en parallèle, tout en permettant d'avoir accès à n'importe quelle ressource à partir de n'importe quelle autre ressource. Le coût principal de cette approche est la surface de silicium. Les TI *rapid-io* actuels offerts par *Tundra Semiconductor* [36] possèdent jusqu'à 16 ports et ils supportent un nombre limité de protocoles de communication. Les prochaines générations vont impliquer plus de ports et de protocoles, ce qui entraîne un goulot d'étranglement plus significatif pour l'ordonnanceur qui décide des connexions et le chemin de transfert des données. D'ailleurs, il est souhaité que l'ordonnanceur prenne sa décision en un faible nombre de cycles afin que des petits paquets hautement prioritaires soient traités rapidement. Des valeurs ajoutées aux TI sont également anticipées comme des fonctionnalités de calculs programmables intégrées.

Dans un tel contexte et avec des défis architecturaux multiples, il est d'une grande importance d'assurer que le TI se comporte de la façon anticipée et qu'il atteint les performances requises avant son implémentation RTL. Pour atteindre ces objectifs, une approche intéressante est la création d'un environnement de validation de performance. Idéalement, cet environnement se doit d'être conçu rapidement en abstrayant certains

éléments d'implémentation non-nécessaires. Cela permet d'éviter l'introduction de délais supplémentaires dans le projet. Il se doit également d'être facilement configurable pour permettre de valider divers scénarios. Il est de plus souhaitable que cet environnement serve de base à la vérification et ainsi permettre la réutilisation du design (« design-reuse » [4], [20]). Les multiples niveaux d'abstraction, la notion de concurrence et les types de données de SystemC, ainsi que toutes les possibilités qu'offre le C++ et les diverses bibliothèques C++ standards, sont d'un grand attrait pour atteindre un tel objectif. Plus le niveau d'abstraction est élevé, plus il est aisé d'abstraire les détails d'implémentation. Cela permet d'avoir un meilleur focus sur l'architecture lors de l'implémentation. La modélisation doit inclure une notion de temps (en termes de cycle) pour bien représenter l'architecture. Ceci est significatif puisque la latence est un des paramètres importants qui permet d'assurer une performance adéquate. Le modèle transactionnel de SystemC (TLM) [30] permet d'atteindre le niveau d'abstraction élevé voulu. Dans le cadre de notre application, le système peut envoyer et recevoir un paquet sous forme de simples transactions. La bibliothèque de SystemC pour la vérification (SCV) permet de rendre disponible des techniques avancées de vérification basées sur les concepts à haut niveau d'abstraction du SystemC et de modéliser divers scénarios [31]. Ces bibliothèques permettent d'obtenir un modèle de haut niveau aisément configurable, ce qui facilite grandement la validation des choix architecturaux. Ils permettent également d'obtenir un environnement qui supporte la vérification.

Il existe des travaux reflétant l'état de l'art pour la validation de TI. Pour la validation de leur propre architecture incluant leur ordonnanceur, Mandiwalla et Tzeng ont effectué des simulations d'implémentations matérielles [21]. Bien que précis, ce type de validation peut nécessiter un effort considérable sans garantir les résultats escomptés. La création d'un modèle adéquat peut améliorer considérablement plusieurs aspects dont le temps de développement, la rapidité d'exécution et la flexibilité de la validation. Dans cette optique de modélisation, on retrouve dans la littérature un environnement qui a été développé à partir de *ns2*, un simulateur de réseau dont le code source est disponible [41].

Bien que *ns2* donne accès à des scénarios de trafic et des statistiques intéressantes sans effort ajouté, il ne permet pas l'aisance de modéliser une architecture au cycle près puisqu'il est axé sur la réseautique. Il n'est également pas aisément intégrable à d'autres étapes de la méthode de conception d'un CI, telle que la vérification. L'environnement développé pour ce projet applique plusieurs concepts d'actualité présentés dans une méthode de vérification avancée d'actualité (« Open Verification Methodology » ou OVM) développée conjointement par Cadence et Mentor Graphics [25]. Ces concepts sont également appliqués pour la validation de performance. Également, XTSC de Tensilica [35] permet d'intégrer un modèle de TI SystemC utilisé dans l'environnement présenté à un système réel incluant un processeur et exécutant une application réelle, ce qui permet une validation et une vérification dans un contexte réel.

La principale contribution espérée du présent projet est de développer une plateforme de validation de performance efficace pour le développement d'un TI. Une des contributions secondaires espérées est d'obtenir une méthode de conception de CI qui, en plus d'intégrer les aspects de la méthode de conception usuelle, telle que la vérification, intègre de façon efficace les concepts de validation de performance développés avec l'environnement créé. Les étapes de validation et de vérification ont potentiellement des éléments communs et ce projet se permet de les exploiter.

L'élément le plus complexe et critique d'un TI est l'ordonnanceur. Il doit rendre la meilleure décision de routage des paquets possible à l'intérieur d'un nombre de cycles petit et souvent fixe qui se doit d'être à une fréquence d'opération la plus élevée possible. Pour atteindre une performance optimale, le TI doit assurer l'équité de ses décisions et une latence faible sur les paquets. L'algorithme d'ordonnancement WRR (« Weighted Round-robin ») [40] est utilisé comme modèle théorique de référence dans ce document, alors que le WWFA (« Wrapped Wave Front Arbiter ») [11] est présenté comme étant une solution intéressante pour une implémentation matérielle. La performance de ces algorithmes est quantifiée pour quelques scénarios de trafic.

Ce mémoire présente, dans le premier chapitre, le contexte et les aspects généraux relatifs au projet. La vérification et validation de performance y sont introduites. L'architecture générale d'un TI ainsi qu'un exemple de banc d'essai simple pour un TI sont aussi présentés. Les diverses statistiques pertinentes au projet suivent. Enfin, ce chapitre aborde les concepts supportés par XTSC (Xtensa SystemC) [35]. Le deuxième chapitre discute de l'environnement de validation de performance et de vérification aisément configurable avec trafic aléatoire contraint réalisé avec SystemC et SCV. Cet environnement permet de rapidement valider des idées architecturales avec un modèle de trafic aléatoire contraint ou spécifique à un comportement particulier. Le troisième chapitre de ce mémoire présente l'ajout de modèles de trafic complémentaires et plus complexes. Une application complète et réaliste y est d'abord présentée. Cette application est sous une forme matériel-logiciel et elle inclut un TI intégré avec XTSC. Il s'agit d'une application réelle de compensation de mouvement [38] qui est exécutée avec un ASIP [2], [3], [18]. Ce même chapitre présente les concepts à utiliser pour incorporer un modèle de trafic existant C++ à notre environnement SystemC, ce qui permet de potentiellement réutiliser des modèles de trafic existants. Cela peut s'avérer important pour une entreprise avec des modèles de trafic existants qui désirent réutiliser les éléments disponibles et éprouvés. Finalement, des résultats de simulation obtenus avec l'environnement décrit au chapitre 2 sont présentés et discutés dans le dernier chapitre. Les trois types de trafic supportés par l'environnement qui sont abordés aux chapitres 2 et 3 sont complémentaires et ils permettent une validation de performance complète et efficace à différents niveaux. Ils s'intègrent bien à une méthode de conception complète, telle que montré au chapitre 4. Le tout est suivi d'une conclusion qui passe en revue les points majeurs de ce mémoire et aborde brièvement les travaux qui pourraient donner suite au projet.

CHAPITRE 1. CONTEXTE ET GÉNÉRALITÉS

Ce chapitre présente divers concepts qui servent de base aux travaux de ce mémoire. Il sert de référence pour le contenu des chapitres suivants. Il traite d'abord divers aspects de la vérification et de la validation de performance. Ensuite, les TI et les statistiques pertinentes aux TI sont discutés. XTSC de Tensilica est également présenté. Finalement, une brève conclusion termine le chapitre.

1.1 Vérification

La vérification se fait en comparant un modèle matériel ou logiciel avec une spécification [4], [27]. Le but est de garantir que le comportement obtenu est bien celui spécifié. Il s'agit d'un aspect important de la conception et il est pertinent si la spécification est adéquate.

Dans un cas réel, comparer un modèle à une spécification est peu pratique. Ainsi, idéalement, une équipe indépendante crée un modèle logiciel à partir de la spécification afin de vérifier un modèle matériel créé par une autre équipe. En ayant deux interprétations de la spécification, cela permet de vérifier une implémentation matérielle. En cas de divergence d'opinion, on doit réviser la spécification pour en retirer l'ambiguïté [4].

La vérification peut se faire de façon formelle [4] ou via simulation. Dans ce dernier cas, un banc d'essai doit être conçu. Il s'agit en fait de l'outil principal pour la vérification. Il existe plusieurs langages pour la vérification et le choix d'un langage influence l'architecture du banc d'essai. L'approche la plus connue et simple est l'utilisation d'un langage de description de matériel, tels que le VHDL et le Verilog. Une autre option est l'utilisation d'un langage de description de matériel étendu.

SystemVerilog est un exemple d'extension du Verilog qui peut être utilisé pour la vérification [29]. Une approche plus classique peut également être réalisée avec des langages logiciels comme le C++ et le Java. Une telle réalisation est présentée à titre d'exemple concret plus loin dans ce chapitre. Enfin, il existe même des langages dédiés à la vérification matérielle, tels qu'OpenVera [4] et SystemC accompagné de SCV [12]. Il s'agit de langages modernes avec un grand potentiel dont la popularité est croissante. L'État de l'art montre que des librairies C++ supplémentaires ont même été créées afin de compléter les fonctionnalités de vérification offertes par SystemC/SCV [26].

Un type de vérification efficace, qui a également servi de base aux travaux de ce mémoire, est l'utilisation d'un langage de vérification de matériel, en l'occurrence SystemC/SCV. SystemC est une bibliothèque de description de matériel et SCV, une bibliothèque de vérification qui s'ajoute à SystemC. Le tout est écrit en C++. Les travaux courants servent d'exploration de ces bibliothèques pour la vérification. SystemC/SCV est une norme établie afin d'uniformiser la vérification et de la rendre portable à plusieurs environnements (C++).

L'un des principaux avantages d'utiliser SystemC/SCV est que cela permet de créer des bancs d'essai à haut niveau d'abstraction rapidement tout en utilisant une bibliothèque C++, donc un langage connu et répandu. Toutes les fonctionnalités du C++ sont donc disponibles, telles que la notion d'orienté-objet, les structures, les chaînes de caractères et la surcharge d'operateurs. Étant du C++, l'exécution est rapide. La rapidité à créer de nouveaux scénarios de test est également un aspect important.

SystemC intègre à ce langage connu la possibilité de travailler avec des types similaires à ceux disponibles dans un langage de description de matériel et il supporte la concurrence d'exécution avec les « sc_module ». Le module est en quelque sorte un composant VHDL. Les « sc_module » s'exécutent en concurrence et communiquent par des ports (« sc_port »). Divers canaux de communication sont disponibles, tels que des

signaux, FIFO, mutex et sémaphores. Les ports des modules utilisent des canaux pour communiquer. La synchronisation des modules est faite soit par événements ou soit par une liste de sensibilité. SystemC possède également les notions de temps et d'horloge.

SCV amène quelques aspects intéressants à la bibliothèque SystemC. Le support de la vérification transactionnelle (TVM [6]) est, sans contredit, l'aspect le plus important. Cela permet d'abstraire, au niveau des transactions, les entrées/sorties du banc d'essai. L'envoi d'un paquet constitue un exemple de transaction. L'abstraction de détails d'implémentation constitue un avantage majeur en vérification, puisque cela permet au concepteur de se focaliser davantage sur la tâche à accomplir, ce qui rend le processus plus efficace. SCV intègre aussi comme éléments majeurs : un générateur aléatoire, des contraintes et poids de génération aléatoire et une fonctionnalité d'observation de variables et transactions.

SCV OSCI est disponible gratuitement. Cependant, la version OSCI permet de simuler du C++ seulement. Donc, elle est parfaite pour vérifier un module SystemC mais sans plus. Cependant, SCV est disponible avec plusieurs simulateurs HDL, tels que NC-Sim et Modelsim, ce qui permet d'effectuer une simulation multi-langage. SCV permet donc d'intégrer des parties de codes diverses écrites en C++ ou VHDL, par exemple. Modelsim impose, par contre, certaines contraintes à l'utilisateur de SCV. Certaines seront discutées au chapitre 2. Modelsim fournit également un ensemble d'outils comme « scgenmod » qui permet de créer de façon automatisée une classe C++ à partir du module HDL. Cette classe inclut la fonction d'accès au module VHDL.

La figure 1.1 montre le modèle transactionnel proposé par SCV pour la vérification. Il est à noter que l'appel au modèle de référence n'est pas spécifié par SCV, mais une implémentation possible est incluse au schéma.

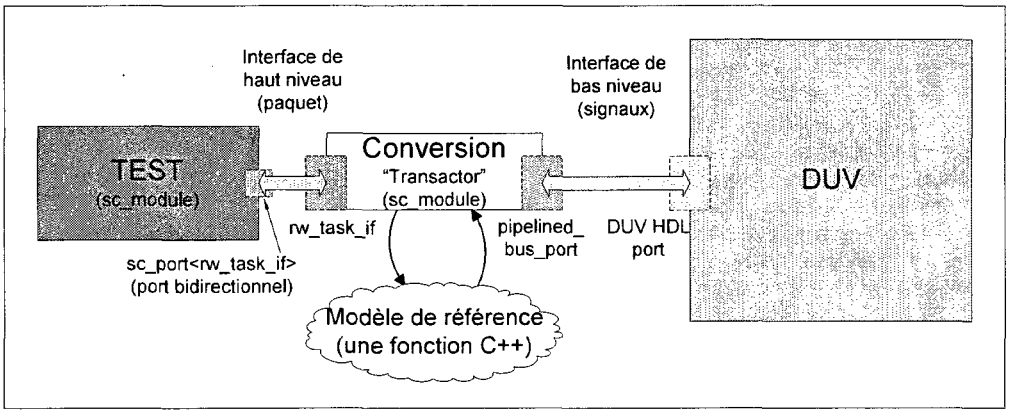

Figure 1.1 Schéma détaillé d'un banc d'essai transactionnel en SystemC

Le banc d'essai contient principalement trois modules : le module de génération de vecteurs de test, le module de conversion de transactions (« transactor ») et le module RTL testé (« Design Under Verification » – DUV). Le module de génération de vecteurs de test contient du code de haut niveau qui génère des transactions en entrées/sorties avec, généralement, des contraintes aléatoires. La conversion sert d'adaptateur entre les transactions du module de test et l'interface matérielle du DUV. Il permet d'abstraire ce détail d'implémentation du module de génération de vecteurs de test. Donc, deux interfaces ont été créées. L'une d'entre elles (« rw_task_if » – voir figure 1.1) définit la structure de l'information transmise (paquet) et les fonctions de lecture/écriture pour l'interface à haut niveau d'abstraction. Cette interface hérite de la classe « sc_interface » de SystemC. La deuxième interface (« pipelined_bus_port ») définit les signaux de l'interface matérielle. Une transaction générée par le module de test serait, par exemple, l'envoi d'un paquet. Le module de conversion de transactions est ensuite en charge de convertir ce paquet, et ce, en plusieurs cycles si nécessaire. Il peut également être utilisé pour abstraire l'appel du modèle de référence du test en temps réel, ce qui permet de cacher cette fonctionnalité au test tout en comparant les sorties au fur et à mesure. Le résultat du modèle de référence peut soit être écrit dans un fichier de sortie pour comparaison ultérieure et/ou comparé en temps réel. Dans les deux cas, il est possible de

produire un résultat global de succès ou échec du test (« pass » ou « fail »). Cela permet de savoir rapidement, après l'exécution du test, si des erreurs sont survenues.

Afin d'illustrer la différence entre la méthode de vérification appliquée avec SCV/SystemC et une méthode plus simple, un banc d'essai écrit complètement en VHDL a été réalisé pour ce projet. Il est présenté à la figure 1.2. Cet exercice permet, entre autres, d'observer les limitations d'une telle approche et d'observer les avantages qu'offre un environnement plus évolué.

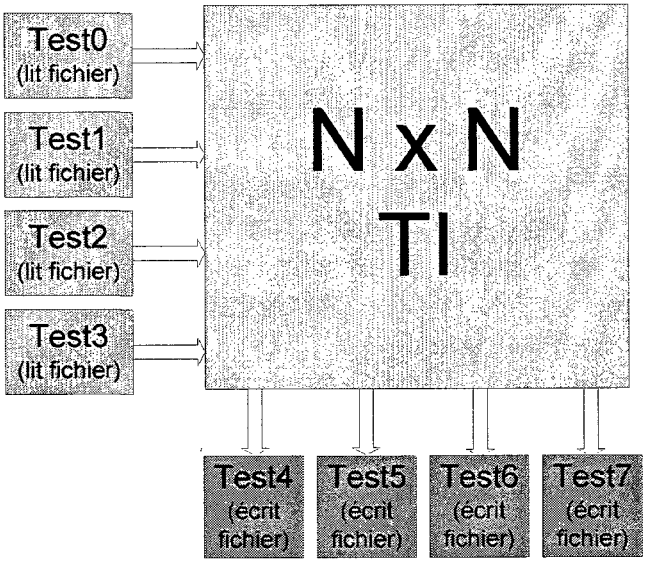

Figure 1.2 Banc d'essai VHDL simple

Il y a un module de lecture des fichiers d'entrée par port. Chacun des modules accède à son fichier propre et se charge de la transmission des données sur les ports d'entrée. Le format du fichier est un ensemble de lignes contenant chacune un mot à envoyer sur le port indiqué sur cette même ligne, suivi d'une autre ligne indiquant si le mot/paquet courant doit être écrit sur le port ou non. Cela permet de modéliser un certain pourcentage d'utilisation de la bande passante de façon simplifiée. Il y a également un

module équivalent pour chaque port de sortie qui produit un fichier de sortie contenant tous les mots transmis du TI avec leur temps spécifique pour chacun des ports.

Bien que la quantité de travail requise pour la création du banc d'essai soit faible, on remarque que celui-ci offre des possibilités très limitées. En effet, il permet de tester rapidement le fonctionnement, mais il est ardu d'obtenir des scénarios de tests spécifiques sans y ajouter des éléments. Avec seulement ce banc d'essai simple, l'usager doit produire manuellement les fichiers qui déterminent les trafics aux entrées, ce qui devient lourd lorsqu'on veut tester plusieurs scénarios de test spécifiques de complexité variable. On voit clairement un besoin pour une architecture plus avancée. Une approche classique pour résoudre cette problématique serait d'ajouter un programme C++ pour la création des fichiers d'entrée lus par le banc d'essai VHDL réalisé. Ce programme, étant en C++, peut utiliser des éléments plus avancés, tels que les objets (un objet complexe de type paquet par exemple), la surcharge d'opérateurs, les pointeurs et les bibliothèques de fonctions qui permettent la génération de nombres aléatoires. Cela permet de rendre le banc d'essai de la figure 1.2 plus efficace et bien plus aisé à configurer avec le niveau d'abstraction plus élevé qu'amène le C++. La simulation d'un test particulier a le désavantage de s'effectuer en deux étapes. Cependant, ce projet amène la méthode de développement d'un circuit à un autre niveau en éliminant ce processus à deux étapes et en y intégrant l'aspect validation de performance. En effet, comme on le verra au chapitre 2, en utilisant une approche basée sur des modules SystemC qui s'exécutent en concurrence, on peut intégrer des modèles de circuits plus avancés. Ceci vient intégrer l'aspect validation de performance du début de projet au processus de conception. Cette validation doit être réalisée pour être efficace de toute manière.

Tel que mentionné, SCV permet de générer des cas de test aléatoires contraints en plus des tests plus conventionnels dirigés et aléatoires. Un test dirigé vérifie un cas particulier. Un test aléatoire vérifie toutes les entrées de façon aléatoire. Un quatrième type pertinent s'ajoute et sera décrit plus en détail au chapitre 3 de ce document; il s'agit

du test appliqué. Nous verrons au chapitre 3 que ce type est également réalisable avec un environnement SystemC/SCV, ce qui en fait un environnement complet. Le tableau 1.1 montre les avantages et inconvénients de chaque type de test.

Tableau 1.1 Avantages et inconvénients de différents types de tests

Type de test	Avantages	Inconvénients
Tests dirigés	- Précis - Utile pour le déverminage ou les cas particuliers	- Long à développer - Laborieux de parcourir un éventail de cas de tests
Tests aléatoires	- Rapidité de développement - Teste rapidement plusieurs bits sur les entrées	- Ne teste pas l'architecture en profondeur - Redondant - Inefficace
Tests aléatoires contraints	- Teste rapidement un grand éventail de cas de test - Adaptation des contraintes de cas de test facilement modifiables - Meilleure couverture - Pas exhaustif mais efficace	- Dans certains cas, on veut tester un cas précis
Tests appliqués	- Permet de tester un cas tiré d'une application réelle	- Environnement plus lourd et moins flexible.

En vérification, plusieurs éléments sont désirables. L'approche SCV en intègre plusieurs. En effet, SCV permet d'avoir un haut niveau d'abstraction ainsi qu'un environnement hautement configurable et auto-vérifiant. De plus, un simulateur comme Modelsim intègre l'aspect couverture (de code, fonctionnelle ou autre [23]) à l'environnement. SCV permet également de séparer l'environnement de vérification des tests. Une approche orientée-objet, telle qu'utilisée en C++, étant modulaire, cela favorise la réutilisation. En effet, on peut facilement s'imaginer créer une bibliothèque de classes

de tests et modules de conversion divers par exemple. Finalement, tout étant codé en C++, il est aisé d'effectuer les comparaisons avec une référence et de produire un résultat de type succès ou échec (« pass » ou « fail »).

Cadence et Mentor Graphics ont maintenant conjointement développé une méthode de vérification moderne, OVM (« Open Verification Methodology ») [25]. Cette dernière provient en fait d'une méthode unifiée de vérification de Cadence (Universal Reuse Methodology (URM) [7], [8]) combinée à des idées de la méthode de vérification avancée (AVM) développée par Mentor Graphics [13]. OVM est disponible avec les simulateurs Incisive® de Cadence et Questa de Mentor Graphics. OVM va plus loin que les standards SystemC et SCV. Il s'agit d'un standard disponible gratuitement qui vise d'abord l'utilisation standard avec n'importe quel simulateur du SystemVerilog pour la vérification. Il propose une bibliothèque normalisée et une méthode de développement éprouvée. Il rend plus aisé la création et l'utilisation de IP préconstruits pour la vérification. Bien que le standard ait été créé principalement pour le SystemVerilog, ces modules VIP (« Verification Intellectual Property ») peuvent être codés en SystemVerilog, SystemC et langage *e*, trois des langages de vérification avancés les plus populaires. L'approche proposée par OVM est disponible depuis 2008 et est une bonne approche à suivre pour un nouveau projet. Le projet courant n'utilise pas cette méthode puisque le standard n'était pas encore diffusé ni disponible au début du projet. Par contre, la plupart des concepts avancés de vérification qui sont inclus dans OVM se retrouvent dans ce projet. Un des éléments concrets que cette méthode aurait apporté au présent projet est d'avoir un environnement indifférent par rapport au simulateur utilisé. Ce projet a été réalisé en suivant les directives de Mentor Graphics non-standardisées pour l'intégration du SystemC. Il s'agit d'un avantage considérable pour OVM, considérant qu'il est souhaitable d'avoir divers modules VIP disponibles ouvertement pour accélérer le processus de vérification et, par le fait même, la validation de performance en appliquant la méthode présentée dans ce mémoire.

OVM se base sur un modèle de vérification transactionnel, tout comme les travaux courants. OVM permet également de modifier aisément des bancs d'essais à l'exécution et d'écrire plusieurs tests à partir de l'environnement de base avec des changements de codes minimaux. Il permet de créer rapidement des tests abstraits du module testé qui peuvent même générer des stimuli aléatoires contraints, aléatoires ou dirigés. Une séquence ainsi créée peut être réutilisée aisément. Il rend disponible également une interface d'écriture de test simple et une méthode d'affichage des messages standardisée.

La Méthode de Vérification Unifiée de Cadence (UVM) [8] présente des concepts intéressants pour la vérification. Ces éléments ne sont pas primordiaux pour ce projet puisqu'ils n'ont aucun impact sur l'aspect le plus important pour ces travaux, la validation de performance. Également, ils ne sont pas pertinents pour le niveau de détail de vérification exploré dans le cadre de ce projet de maîtrise. Par contre, il est intéressant de pouvoir appliquer ces éléments disponibles avec la plupart des simulateurs avancés à une phase ultérieure de vérification. Il s'agit, entre autres, du concept de couverture autant du code, que structurel, ainsi que de l'application et des interfaces. La réutilisation des éléments d'un banc d'essai est également primordiale pour améliorer le temps de développement. La méthode d'interface transactionnelle présentée s'applique également au niveau modulaire. En effet, une bonne approche modulaire consiste à créer plusieurs modules du design avec une interface transactionnelle. Au départ, tous ces modules communiquent directement au niveau de la modélisation transactionnelle. En cours de projet, des modules matériels peuvent être intégrés graduellement avec les modules de conversion des transactions adéquats (« transactor »). L'aspect rapidité d'exécution avec une méthode d'accélération est également un aspect important. Le prototypage FPGA ou l'utilisation d'un émulateur matériel permet d'accélérer les simulations. D'ailleurs, cette accélération peut se faire de façon modulaire en intégrant un modèle matériel avec d'autres modèles logiciels à haut niveau et les modules de conversion adéquats

(« transactor »). Il n'est donc pas nécessaire d'avoir un système entier pour réaliser une simulation matérielle rapide.

1.2 Validation de performance

La validation de performance se fait en comparant un modèle avec les requis. Il permet d'évaluer si le modèle atteint les performances voulues. Cela peut se traduire en une évaluation de la performance d'une architecture avec un modèle sommaire qui inclut les caractéristiques importantes pour permettre une validation. Cette évaluation est comparée aux requis selon certains critères établis. Il s'agit d'un paramètre distinct de la vérification qui a pour but de vérifier que le comportement obtenu correspond bien à la description contenue dans la spécification. La validation doit idéalement être faite en début de projet sur un modèle en cas d'incertitude sur les performances, et, également, une fois l'implémentation matérielle complète pour confirmer l'atteinte des performances recherchées. On voit que, théoriquement, il est possible d'avoir un seul environnement qui applique les stimuli sur un modèle et sur une implémentation matérielle. Le chapitre 2 développe cette idée.

La validation en début de projet est un aspect à prioriser en cas d'incertitude pour le projet. Le coût du temps passé à faire cette validation est nettement inferieur au coût d'un projet complété qui n'atteint pas les performances désirées. Par contre, trop de temps passé à la validation peut avoir un impact sur l'échéancier du projet. Il faut par conséquent bien doser le temps passé à réaliser des tâches de validation. Il est donc essentiel d'avoir une méthode de validation efficace.

La littérature propose diverses méthodes pour valider la performance. Des travaux publiés par Armstrong, Gray et Vuppala montrent une méthode de validation qui se base sur la création d'une modélisation simplifiée en VHDL [1]. La littérature prouve également qu'une simulation SystemC s'avère significativement plus rapide qu'une

simulation RTL [9]. En plus du haut niveau d'abstraction que le SystemC offre, cette constatation en fait un candidat idéal pour modéliser une architecture dans le but d'analyser sa performance. On s'attend ainsi d'obtenir un temps de conception du modèle et de simulation amélioré. L'état de l'art montre quelques exemples qui utilisent cette approche afin de créer un environnement de validation pour l'exploration architecturale de CI divers [22].

1.3 Tissu d'interconnexion

Un tissu d'interconnexion (TI) [28] peut se définir, de façon générale, comme étant une structure de communication qui reçoit des paquets sur des ports d'entrée et les transfère vers les ports de sortie. Un modèle de base est présenté à la figure 1.3. Le tissu d'interconnexion est surtout utilisé pour faire la communication entre divers modules passifs et actifs dans un système embarqué. Les modules peuvent être un processeur avec jeu d'instruction spécifique (ASIP) ou un FPGA/ASIC dédié qui accède à plusieurs mémoires ou ports de communication, comme un port Ethernet. D'autres unités de calculs peuvent opérer en parallèle et accéder aux mêmes mémoires au travers du tissu d'interconnexion. Les paquets transmis pour ces applications sont de tailles variables contrairement aux TI utilisés pour le routage de cellules (« cell based routing »), comme celles utilisées dans les systèmes ATM. Ce dernier type de communication n'est pas traité dans le présent ouvrage. Il est à noter que l'interconnexion à granularité de cellule est un cas particulier qui peut aisément être couvert par ces travaux en supposant des paquets de tailles égales, que ce soit en convertissant le protocole en entrée/sortie du medium ou en ne supportant qu'un protocole de paquet à tailles fixes de façon native. Une architecture de type TI permet d'avoir plusieurs communications indépendantes en parallèle, contrairement à une architecture de type « bus », qui ne permet pas les communications simultanées. En effet, pendant une communication, le medium est bloqué pour toutes les autres transmissions et ce, même si les communications sont totalement indépendantes

[37]. Cela constitue l'avantage principal du TI au prix d'un médium d'interconnexion plus complexe et couteux en termes de surface de silicium.

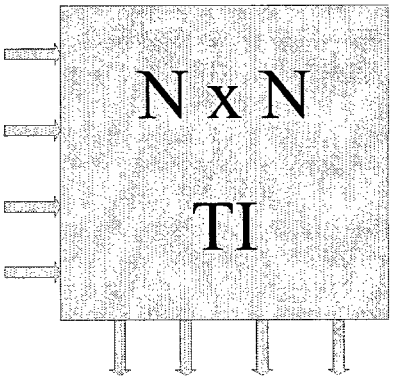

Figure 1.3 Représentation de base d'un tissu d'interconnexion

Il existe plusieurs types d'architectures qui ont chacun leurs avantages respectifs selon le type de protocole de communication utilisé. Pour les travaux courants, une architecture connue pour son efficacité dans des systèmes embarqués a été utilisée. Il s'agit d'une architecture de type commutateur crossbar [28], [37] qui utilise des files de sortie virtuelles (« Virtual Output Queuing » – VOQ) [34]. Les files sont utilisées en entrée. Pour chaque entrée, il existe une file par sortie. Dépendamment de la destination du paquet en entrée, celui-ci est placé dans la file de sortie virtuelle correspondante. Cette architecture permet de faire une classification des paquets dès leur arrivée dans le TI. Cela permet aussi de préserver l'ordre des paquets et d'éviter le blocage de file par le premier élément (« head of line blocking »). Ce dernier événement se produit lorsqu'un paquet à la tête de la file bloque un autre paquet de cette file qui serait en mesure d'être transmis dans le TI. Puisque les paquets sont classifiés dans des files virtuelles séparées, cela ne peut se produire avec une architecture qui utilise des VOQ. La figure 1.4 montre l'idée de base de ce type d'architecture.

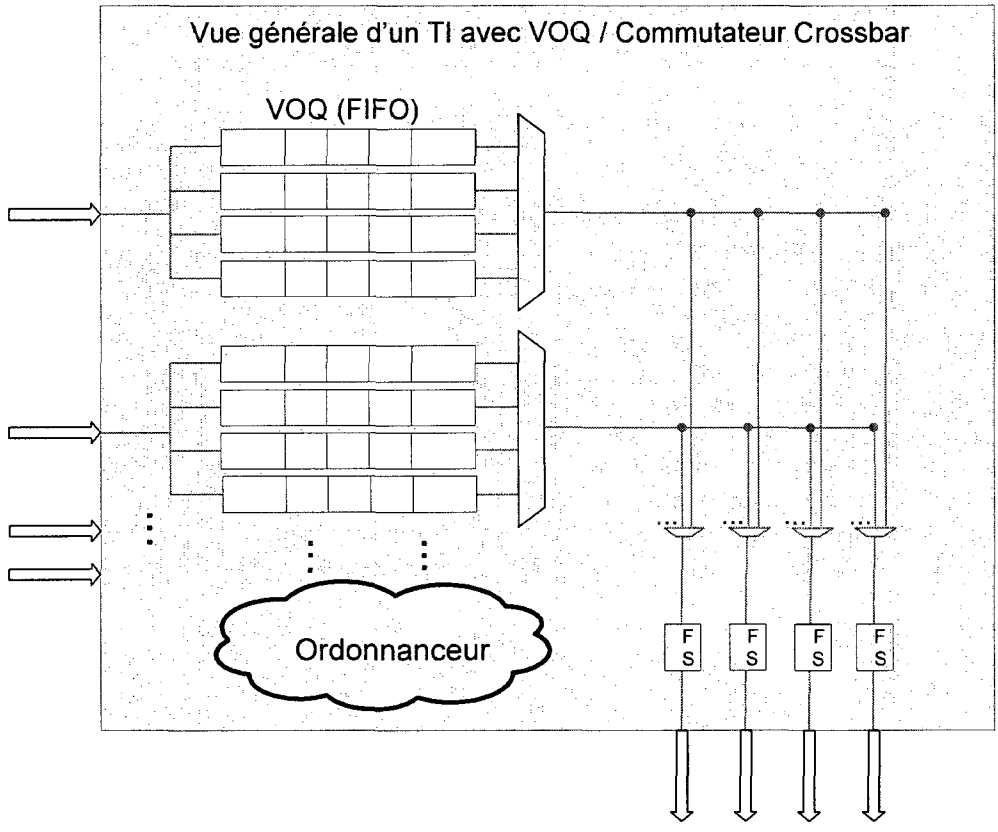

Figure 1.4 Architecture de base d'un tissu d'interconnexion de type commutateur crossbar à files de sortie virtuelles

Chaque port d'entrée possède son propre ensemble de files de sortie virtuelles. Ces files sont dites virtuelles car, ayant besoin d'un seul port d'écriture/lecture au FIFO par port d'entrée du TI, l'implémentation est habituellement faite avec une seule mémoire qui est gérée avec des pointeurs. Pour des profondeurs de files considérables, cela devient avantageux en termes de surface de silicium parce que la mémoire est utilisée de façon plus efficace. Une fois le paquet dans une des files virtuelles, sa destination étant connue, il ne reste qu'à arbitrer les diverses files s'adressant à la même sortie. Les files de sortie virtuelles font ensuite suivre les paquets vers le commutateur crossbar un à la fois. Le commutateur crossbar [28] se définit comme étant le chemin de données qui route les

paquets vers les sorties. Il suit les files dans le chemin de données. Il s'agit souvent d'un agencement de multiplexeurs (tels que montrés à la figure 1.4) et de fils habituellement longs puisqu'ils doivent parcourir physiquement presque toute la hauteur ou la largeur de la puce de silicium. Avec un commutateur crossbar, une entrée peut envoyer un seul paquet à une seule sortie à la fois. Cela constitue un compromis afin de simplifier la circuiterie et les contraintes. Un commutateur crossbar peut être accéléré par rapport aux ports d'entrée/sortie en ayant un débit plus important. Le rapport de bande passante entre les chemins de données du commutateur crossbar et ceux des ports d'entrée/sortie est définit comme le facteur d'accélération. Habituellement, il existe un étage de files de sortie (FS) qui contient une quantité minimale de mémoire tampon avant que le paquet ne soit envoyé à la sortie du tissu. Cela permet, entre autres, d'ajuster la largeur des données à la largeur du port de sortie et d'absorber les irrégularités de débit du TI. Combiné au facteur d'accélération du commutateur crossbar, la mémoire tampon en sortie permet d'améliorer les performances du TI en diminuant les contraintes sur le matériel. L'ordonnanceur est, en quelque sorte le chef d'orchestre qui choisit quels paquets sont envoyés au commutateur crossbar et à quel moment. Il utilise habituellement une forme de logique pour prioriser les divers paquets à partir des informations des diverses files. Par exemple, si aucune des files n'est vide, alors toutes les entrées tentent de transmettre un paquet vers toutes les sorties. Différents mécanismes pour la gestion des priorités peuvent être utilisés en fonction de paramètres comme : le temps que le paquet a passé dans la file, la longueur et le type des paquets, leur priorité. Tous ces facteurs tendent à rendre l'ordonnanceur plus complexe, mais aussi plus efficace. Cependant, il faut tout de même définir un mécanisme de priorité dans le cas ou chaque connexion entrée/sortie a un poids équivalent. Cette priorité se doit d'être variable dans le temps afin d'assurer une décision équitable. Chaque connexion entrée/sortie doit avoir la même chance d'être sélectionné. La priorité est donnée par l'algorithme à chaque appel de l'ordonnanceur, mais en moyenne, après un certains temps, la décision se doit d'être équitable pour être efficace. Un algorithme efficace demeure équitable même si les paquets d'entrée ne sont pas de tailles égales entre les ports.

Une version simple et efficace d'ordonnanceur est le « weighted round-robin » (WRR) [28], [40]. Le WRR étant un algorithme théorique, cela en fait une référence intéressante pour ce projet. Pour cet algorithme, la priorité est donnée en alternance à la manière d'un « round-robin » entre les files. À chaque décision, la plus haute priorité est donnée à un port de sortie et une alternance « round-robin » est appliquée sur le port prioritaire pour la décision suivante. La priorité est initialement, et sans perte de généralité, donnée au port 0. Il en est de même pour le port d'entrée prioritaire mais la priorité passe au port d'entrée suivant seulement après que tous les ports de sortie aient reçus la plus haute priorité pour un port d'entrée donné. L'algorithme inclut également une notion de poids qui se doit d'être fixe et de refléter le trafic en entrée. Pour un ordonnancement, l'entrée qui nécessite une plus grande bande passante (en termes de paquets) selon ce poids se voit assigner plusieurs paquets. Par exemple, un port d'entrée qui reçoit des paquets deux fois plus petits transmettrait deux paquets lorsqu'il est choisi et que ces deux paquets sont disponibles. Étant donné que le trafic est défini dans notre environnement et que le trafic à chaque entrée est connu, il est facile de configurer le poids de chaque entrée. Ce n'est généralement pas le cas dans un système réel, mais cela donne un avantage lors d'une analyse. Cet algorithme sert en fait de référence pour une analyse de performance avec n'importe quelle architecture d'ordonnanceur. L'algorithme est théoriquement efficace mais n'est pas réalisable physiquement si le nombre de ports est élevé. Ceci vient du fait que le nombre de connexions potentielles à analyser en un nombre de cycles fixe croît en fonction du carré du nombre de ports du TI. Donc, avec un grand nombre de connexions à analyser en un nombre de cycles faible, le nombre de niveaux logiques nécessaires à l'analyse décisionnelle devient trop élevé, ce qui rend la fréquence d'opération du circuit peu intéressante. Le nombre de cycles pour la décision se doit par ailleurs d'être faible afin d'éviter de dégrader la performance pour les petits paquets. Ces derniers sont reçus rapidement et ils doivent donc idéalement être transmis rapidement. Une autre approche doit être envisagée pour l'implémentation.

Le WWFA (« Wrapped Wave Front Arbiter ») [11], en plus d'avoir un potentiel de performance intéressant, est beaucoup plus adapté à une implémentation matérielle. Il se prête bien au contexte de ce projet puisqu'on s'y intéresse pour d'autres travaux avec *Tundra Semiconductor* [36]. Cet arbitre permet d'améliorer la fréquence d'opération du circuit en distribuant la décision parmi les nœuds du TI. La figure 1.5 montre les quatre vagues diagonales utilisées par l'algorithme pour un TI à quatre ports. Chaque diagonale contient une seule fois chacun des ports de sortie. Il en est de même pour les ports d'entrée. Ceci est l'élément qui permet de distribuer la décision. À chaque décision, la diagonale prioritaire est alternée. Chacun des nœuds communique à ses nœuds voisins si la connexion est établie ou non. Donc, chaque nœud de la diagonale prioritaire influence les autres nœuds en propageant sa décision. Il en est de même pour les nœuds voisins.

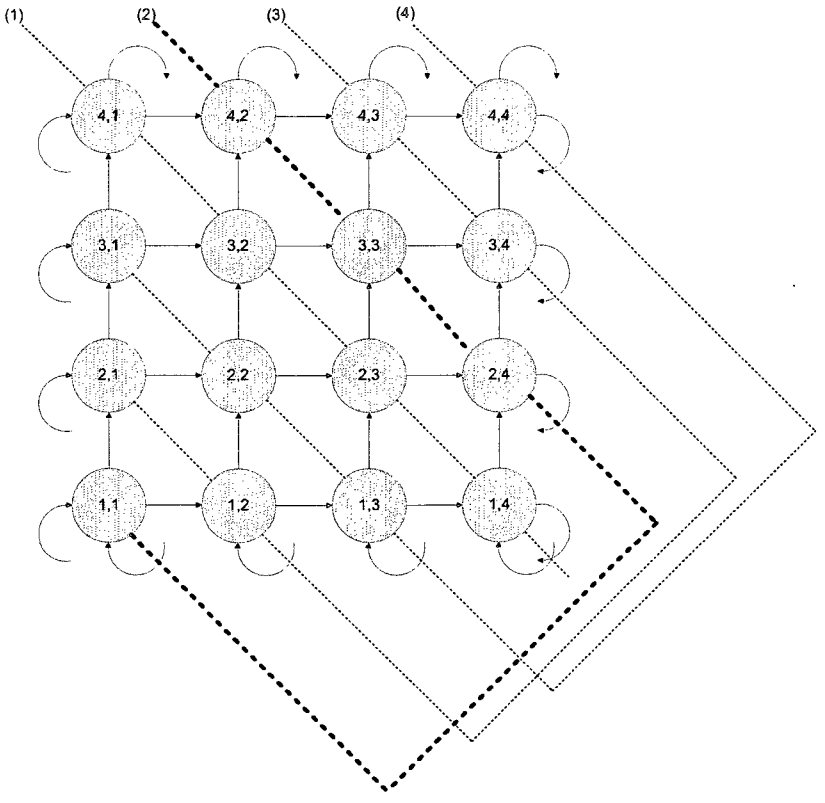

Figure 1.5 Algorithme d'ordonnanceur WWFA ("Wrapped Wave Front Arbiter")

1.4 Statistiques pertinentes aux tissus d'interconnexion

Un des paramètres de base de toute interface de données en matériel est le débit de données, soit la bande passante. Ce paramètre peut être quantifié de diverses façons. La plus commune est de présenter le débit en bits par seconde. Une abstraction plus élevée de ce paramètre est le pourcentage d'utilisation d'un lien. Celui-ci abstrait le nombre de bits passés, mais est valable lorsqu'on compare deux liens de même largeur par exemple. Ces deux descriptifs du débit permettent de quantifier les débits de divers ports d'un TI.

La latence est un autre paramètre important de tout système matériel. Dans le cas d'un TI, on parle de latence d'un paquet dans le TI. Il s'agit du temps qu'un paquet a passé dans un TI avant d'en être sorti. Pour obtenir une comparaison juste, on considère pour ce projet que la latence se calcule entre le moment où le paquet est complètement arrivé dans le TI et le moment où il en est complètement sorti. Avec cette convention, la latence minimale obtenue est le nombre de cycles que prend le paquet sur le lien d'un port. La latence moyenne sur un nombre significatif de paquets est évidemment influencée par les débits aux entrées et l'architecture du TI.

L'équité de l'ordonnanceur est un paramètre pertinent aux TI. L'ordonnanceur décide, en cas de conflit, à quel port donner la priorité. L'équité définit une mesure de la qualité de l'arbitrage entre les divers ports d'un TI. Il existe plusieurs publications [15], [19], [33] qui comparent ou évaluent l'équité de différents ordonnanceurs avec différentes métriques. L'analyse peut être pour un trafic équilibré ou non. Plusieurs scénarios sont possibles. En fait, contrairement à la latence ou au débit, ce paramètre n'a pas de formule préétablie. Plusieurs mesures d'équité sont utilisées pour ces travaux et elles sont décrites ici. Selon quelques sources, il peut être calculé en évaluant le $99^{\text{ième}}$ percentile de la latence [32], [33]. Pour un même trafic global, un TI plus équitable devrait donner un $99^{\text{ième}}$ percentile de latence plus petit. Une autre façon de le mesurer est de comparer les latences moyennes pour les divers ports pour un certain trafic connu

[15]. Il y aussi la comparaison du nombre de paquets perdus par port et du taux d'utilisation des files VOQ par port qui permet d'indiquer l'équité d'un TI suite à une simulation. On s'attend à ce que ces trois derniers paramètres discutés soient équivalents pour tous les ports d'entrée en présence d'un ordonnanceur équitable. Pour des fins d'analyse, une métrique non-standard a également été utilisée. Il s'agit de l'équité de connexion. Elle se définit comme étant le rapport du nombre de cycles d'envoi de paquets pour une entrée donnée alors qu'il y a au moins un paquet à envoyer de cette entrée. Les cycles pour lesquels aucune donnée n'est à transmettre sont ainsi éliminés de l'analyse. Ce rapport peut être comparé pour les diverses entrées.

Puisque ces mesures d'équité mènent à une analyse d'uniformité, il est intéressant d'avoir une métrique qui évalue cette uniformité pour un certain nombre de valeurs. On peut vouloir, par exemple, calculer l'uniformité de la latence par port pour un TI. Une façon intéressante de le calculer est avec le critère d'équité établie par Jain [17]. Voici le calcul théorique pour ce paramètre de Jain.

$$F = \frac{(\sum_{i=1}^{n} y_i)^2}{n\sum_{i=1}^{n} (y_i)^2}$$

Où, dans notre cas, y_i est l'un des paramètres d'équité décrit ci-haut pour le port i, n représente le nombre de ports et F représente le facteur de Jain pour le paramètre choisi. Ce facteur peut prendre des valeurs entre 0 et 1, 1 représentant une grande équité. Pour un trafic non-équilibré, dans le cas où on connaît les trafics en entrée, on s'attend à ce que certaines entrées, pour certains paramètres, aient un poids plus ou moins élevé. Ainsi, pour un algorithme juste, un port avec une taille moyenne des paquets deux fois plus grands qu'un autre port avec trafic similaire devrait perdre deux fois moins de paquets, ce qui correspond à une même quantité de données. Une équation proposée par Jain qui mesure l'équilibre en tenant compte de poids par port w_i s'énonce comme suit:

$$F = \frac{(\sum_{i=1}^{n} w_i y_i)^2}{n\sum_{i=1}^{n} (w_i y_i)^2}$$

Pour notre exemple de paquets perdus précédent, on pourrait choisir w_i à 1 pour un port qui a des paquets deux fois gros qu'un autre port avec w_i à 2. Une autre publication présente un concept similaire pour une application différente (TCP) [24].

1.5 XTSC de Tensilica

La compagnie Tensilica offre des ASIP qui sont des processeurs génériques que l'usager peut configurer afin d'ajouter de nouvelles instructions spécialisées pour une application donnée [2], [3], [18]. Ceci se fait par la description de la fonctionnalité des instructions ajoutées dans un langage matériel propre à Tensilica, le TIE. Cela permet de créer un processeur qui, en plus de posséder tous les atouts d'un processeur générique incluant la stabilité des divers modules fixes, intègre une partie configurable pour spécialiser le processeur. Tensilica fournit un simulateur pour ce processeur configuré et peut également fournir une version matérielle du processeur. Un code C s'exécutant sur le processeur spécialisé fait un appel explicite à l'instruction ajoutée pour en tirer profit. Étant un environnement réel permettant d'exécuter une application tout aussi réelle et étant disponible dans une version SystemC (XTSC), cet environnement devient pertinent dans le cadre d'un projet de vérification et validation avec SystemC/SCV. Il permet d'avoir des scénarios d'entrée/sortie tirés d'une application réelle sans avoir à créer à partir de zéro un nouveau système pour chaque nouvelle application.

XTSC inclut des modules de Tensilica contenus dans des « sc_module ». Entre autres, xtsc_core est un module SystemC qui inclut le simulateur d'instructions de

Tensilica. Il inclut des méthodes qui permettent l'accès aux divers ports : les ports de la mémoire locale, l'interface de la mémoire locale de Xtensa (XLMI), l'interface processeur (PIF), l'interface d'instruction TIE et certains ports d'entrée/sortie du système. Cet objet permet de charger un programme à exécuter et plusieurs autres paramètres de configuration et fonctionnalités. Le fonctionnement des interfaces XTSC TLM sont décrites en détail à la section 3.1 du guide de l'usager XTSC [35]. Tensilica fournit son propre jeu d'outils. Il inclut, entre autres, un outil de simulation (xtsc_run) et un outil de déverminage (xt-gdb).

Il est donc possible pour l'usager d'interfacer n'importe quel module SystemC avec le processeur Tensilica via diverses connexions par des appels de fonctions. On peut ainsi simuler ce module SystemC avec l'environnement fourni par Tensilica.

1.6 Conclusion

Ce chapitre permet de prendre connaissance des aspects généraux de la vérification, de ce qu'amène un langage pour la vérification avancée, des principaux aspects architecturaux d'un TI, des statistiques de performance pertinentes aux TI et de la possibilité d'intégrer un trafic tiré d'une application réelle qui s'exécute sur un processeur. Tous les éléments sont maintenant en place pour présenter un environnement efficace qui intègre la validation de performance et la vérification pour un TI.

CHAPITRE 2. ENVIRONNEMENT DE VALIDATION/VÉRIFICATION À HAUT NIVEAU

Bien que le chapitre 1 décrive plusieurs aspects pertinents au projet, ces derniers ne constituent pas l'essence du projet. Un des aspects importants du projet est de développer un environnement à haut niveau d'abstraction pour la validation et la vérification rapide et efficace des TI. Un des objectifs secondaires est d'intégrer efficacement la validation de performance à la méthode de conception d'un CI. Le présent chapitre montre un tel environnement et comment il s'intègre bien à une méthode de conception complète.

2.1 Concepts et description de l'environnement de validation

L'environnement de validation de performance proposé dans ce chapitre est présenté dans cette section. Il opère à un haut niveau d'abstraction et est aisément configurable.

2.1.1 Généralités

Un des buts principaux du projet est d'être en mesure, à partir d'un environnement de simulation, de valider diverses architectures en les soumettant à diverses contraintes de trafic. Afin de permettre la validation aisée et rapide de diverses implémentations et de pouvoir avoir un grand contrôle sur les scénarios de trafic, il est préférable d'avoir un environnement qui intègre les caractéristiques essentielles au comportement d'un système, mais qui fait abstraction de certains éléments d'implémentation. Par exemple, l'aspect d'implémentation matérielle, de fréquence d'opération, de surface de silicium et toutes les complexités qui en découlent, peuvent

être mis de coté le temps d'une analyse architecturale. Ces analyses architecturales sont habituellement faites en début de projet. Le véritable enjeu est de connaître l'efficacité d'une idée architecturale sans avoir à y passer des mois de travail d'implémentation et de vérification qui seront potentiellement mis au rencart. Pour ce faire, différents langages de programmation et outils de conception peuvent être envisagés. Il est évidemment impossible de couvrir toutes les solutions possibles dans le cadre d'un travail de maîtrise.

Une des avenues les plus prometteuses est approfondie ici. Il s'agit du langage C++ qui est une approche répandue dans l'industrie lorsqu'il s'agit de reproduire le comportement d'un circuit intégré. Pour une analyse de performance comme celle d'un tissu d'interconnexion (TI), un langage de haut niveau comme le C++ a clairement un point faible : il n'inclut pas la notion de temps et de cycle d'horloge. Cependant, la venue de bibliothèques comme SystemC et SCV permettent d'élargir les possibilités du C++. Le grand avantage dans notre contexte est la possibilité d'intégrer une notion de cycle à un modèle de haut niveau C++ tout en conservant la possibilité de codage de haut niveau que permet ce langage. La simulation avec une telle bibliothèque nous donne un avantage au détriment du temps de simulation. Ceci vient du fait qu'à chaque intervalle de temps fixé par le pas de simulation, il faut évaluer les divers processus concurrents, peu importe s'ils auront un impact ou non sur une variable ou un objet quelconque. Avec un code C++ sans notion temporelle ni concurrence des processus, l'évaluation ne se fait que lorsqu'un objet ou une fonction spécifique est appelée sans tenir compte d'un temps de simulation. Il faut être conscient de cette limitation afin de prendre la bonne décision relative aux besoins pour une exploration architecturale. Il est important de choisir l'approche la plus légère qui répond aux besoins spécifiques du projet afin d'accélérer l'implémentation et la simulation au maximum. Notre premier objectif étant d'obtenir un modèle rapidement, tout en ayant la notion de cycle pour modéliser les transmissions de paquets et l'implémentation matérielle de façon réaliste, l'option du SystemC est toute désignée.

L'environnement de validation de performance réalisé est constitué d'un module SystemC principal qui inclut tout ce qui est relatif au TI. Pour ce qui est de la génération du trafic, elle est constituée de *N* modules de test pour *N* ports bidirectionnels (ou 2N ports unidirectionnels) qui permettent d'établir le scénario de trafic de façon indépendante sur chacun des ports. Les modules décrits sont développés en ayant en tête de rendre modulaire les concepts et caractéristiques et de rendre le tout configurable. Un fichier de configuration est modifié avant chaque simulation afin d'établir les contraintes de trafic et d'architecture désirées. Une fois un type de trafic réalisé et une idée architecturale implémentée, il est aisé de rendre ces derniers disponibles via le fichier de configuration.

Étant un environnement à relativement haut niveau d'abstraction, l'interfaçage et le traitement interne au tissu se fait en termes de transactions. C'est-à-dire que, malgré la granularité du système (en cycles d'horloge), les communications et le traitement interne se font en termes de transactions relativement complexes, en l'occurrence, des paquets. Ceci permet de grandement simplifier l'implémentation et de garder le focus sur la tâche à accomplir. Un paquet reçu est analysé d'après sa longueur et un nombre de cycles de transmission est calculé à l'interne. Ce nombre de cycles est appliqué lors des écritures/lectures aux files de même que lors de la transmission dans le commutateur crossbar. L'essence du module TI est tout de même évaluée à chaque cycle. À chaque cycle, les compteurs internes de nombres de cycles restants à la transmission sont décrémentés. Des écritures/lectures suivent selon l'état des compteurs.

La communication entre les modules de test et le module de TI peut se faire de deux façons. Il peut s'agir d'une interface bloquante ou non-bloquante. Les deux implémentations ont été réalisées, mais l'approche bloquante a été retenue au final pour la configuration présentée dans cette section, car celle-ci représente mieux la réalité. Le module de test n'a pas à se soucier des temps de transmission et le code s'en trouve simplifié. Ceci permet d'abstraire complètement la notion de temps du test. Dans le cas

de l'implémentation non-bloquante, il y a deux options possibles. La première est que le module de test et le module de TI simulent tous les deux le temps de transmission d'un paquet sur le port afin de se synchroniser. L'autre option est que le module de test interroge le TI sur la disponibilité du port. Comme il sera montré au chapitre 3, cette dernière approche a été prise pour créer un module de conversion d'interface non-bloquant (*if. non-bloquante*). Ce module est nécessaire pour les besoins de l'environnement présenté à la section 3.2, mais il peut également être utilisé aux interfaces selon les besoins spécifiques de l'utilisateur de l'environnement.

La figure 2.1 présente une vue sommaire de l'environnement de validation de performance. Chacun des modules de test décrit le comportement du trafic par port. Un module « moniteur » est de plus connecté entre chacun des tests et le TI afin de créer une base de données décrivant les transferts de paquets.

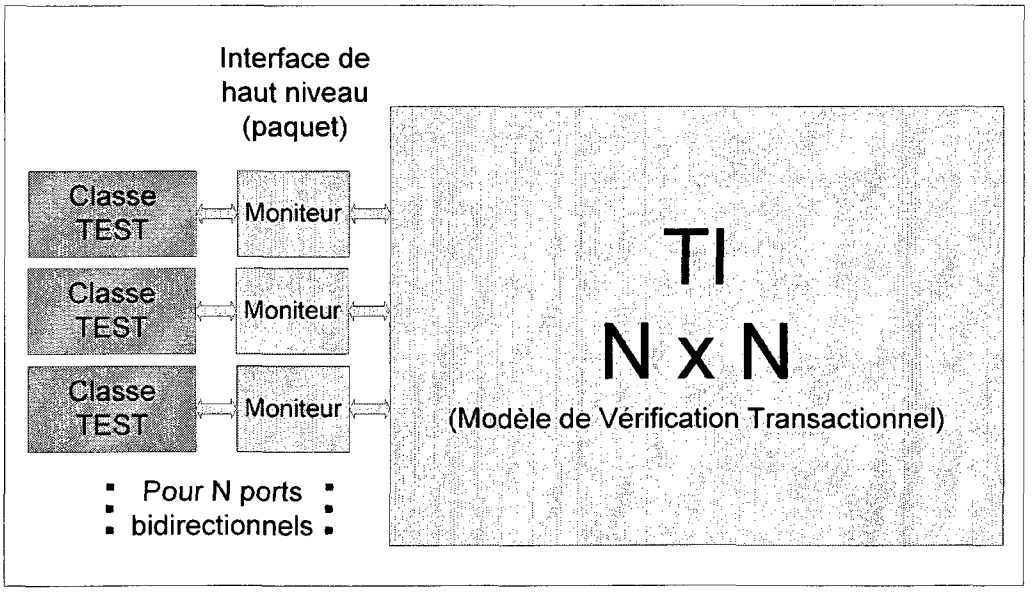

Figure 2.1 Vue d'ensemble de l'environnement de validation de performance

2.1.2 Description du modèle de paquets et des opérateurs de paquets

Le tableau 2.1 représente la structure des paquets.

Tableau 2.1 Description du paquet

Champ	Taille (bits)	Description
dest_port	ADDR_W	Adresse (port) destination du paquet (0 à N)
src_port	ADDR_W	Adresse (port) source du paquet (0 à N)
packet_id	PACK_ID_W	Sert à identifier le paquet par un ID unique (par connexion d'entrée/sortie)
payload	PAYL_W	Sert à identifier un paramètre propre au paquet qui pourrait être modifié à l'interne du TI (latence, QOS, ...)
packet_length	DATA_WD_W	Longueur du champ de données (en octets)
data	Variable	Données pour le paquet courant

L'identificateur de paquet (« *packet_id* ») par connexion sert à identifier de façon unique et simple un paquet donné. Cette technique est valide pour la simulation tant que le nombre de bits alloués pour l'identificateur est suffisant pour ne pas permettre à deux paquets avec un même identificateur de coexister en même temps dans le TI. Pour permettre une analyse statistique complète et sans ambigüité, chaque identificateur doit être utilisé une seule fois par connexion pour une simulation donnée. La taille des données étant variable en fonction d'un des paramètres de taille fixe (« *packet_length* ») déterminé lors de l'exécution à partir des contraintes de génération aléatoire, il est pertinent d'utiliser une allocation dynamique. Différentes implémentations sont possibles. Pour ce projet, les données sont représentées avec une chaîne de caractères allouée dynamiquement à la création du paquet une fois la taille de celui-ci connue. Une chaîne de caractères ASCII limitée aux nombres hexadécimaux a été choisie afin de rendre plus simple l'impression des données en hexadécimal. L'allocation dynamique a lieu après la génération aléatoire de la taille du paquet par l'appel d'une fonction de création des données. L'espace mémoire est alloué avec l'opération C++ « new » dans cette fonction.

Ensuite, chacun des caractères ASCII est généré avec comme contrainte qu'il doit être choisi parmi les chiffres hexadécimaux (i.e. entre 0 et F). Il est à noter qu'un champ de données complet n'est pas absolument nécessaire pour l'environnement de validation, mais il est pertinent lors de la phase ultérieure de vérification.

Le « payload » est un élément générique qui a été inclus sans toutefois avoir été utilisé au cours du projet. Il est disponible pour des travaux futurs. Il représente une caractéristique ou une statistique relative au paquet traité qui peut être modifié à l'intérieur du TI. Il pourrait être un nombre de cycles d'attente ou un niveau de priorité variable dans le temps pour un même paquet, par exemple. Dans un tel cas, ce niveau de priorité aurait un impact sur l'ordonnancement.

L'utilisateur est libre d'ajouter les champs nécessaires puisqu'il est relativement simple de modifier la description du paquet. Le tout est contenu dans quelques classes utilisées uniquement pour décrire les fonctionnalités du paquet. Seuls les nouveaux éléments du paquet servant une fonctionnalité particulière dans le TI influenceront l'implémentation à l'extérieur des classes de description de paquets. Cela montre la modularité amenée par le C++. Un exemple de champ pertinent à ajouter pourrait être un niveau de priorité fixe (ce qui est différent de ce qui a été décrit au paragraphe précédent). Il pourrait aussi y avoir un champ « channel » servant à transmettre diverses données vers une même destination mais qui se recoupent dans le temps. Ceci est utilisé pour transmettre avec les protocoles RapidIO, PCIe et Ethernet.

Ayant une structure de paquet utilisée à profusion, il est également souhaitable d'avoir quelques fonctions ou surcharge d'opérateurs qui seront utilisées régulièrement. Ainsi, l'operateur d'insertion « << » a été créé pour permettre d'afficher aisément un paquet. Également, les operateurs de test d'égalité et d'inégalité ont été créés pour des fins de vérification. Ils permettent de vérifier l'égalité des paquets lors d'une comparaison entre la sortie d'un modèle et celle d'une référence. L'opérateur

d'assignation, qui permet de copier un paquet, est aussi très utile. Entre autres, ce dernier appelle dynamiquement la fonction de création du champ de données.

Cet usage de création dynamique et de surcharge d'opérateurs utilise en fait le C++ comme levier pour rendre l'environnement de validation efficace et simple à utiliser.

2.1.3 Description des paramètres de configuration architecturale du TI

Le tableau 2.2 montre les divers paramètres configurables inclus dans un fichier de configuration pour le TI SystemC utilisé pour la validation de performance.

Tableau 2.2 Description de configuration du TI SystemC (haut niveau)

Champ	Description
Type d'ordonnanceur	WRR ou WWFA
Nombre de ports du TI	Taille du TI
Facteur d'accélération du commutateur crossbar	Facteur d'augmentation de la bande passante dans le commutateur crossbar du TI par rapport aux entrées/sorties
TI SystemC vs RTL	Utiliser le TI SystemC ou matériel
Poids par port pour l'ordonnanceur WRR	Poids du trafic par port utilisé par l'ordonnanceur WRR
Profondeur des files VOQ	Profondeur des files VOQ en termes de paquets avec MUL_FACTOR_BW_DL à 1 (voir plus bas)
Profondeur des FIFO de sortie	Profondeur des FIFO de sortie en termes de paquets
Largeur des ports	Nombre de bits par port
Période d'horloge	Temps d'un cycle de l'horloge (en ns).
Nombre de cycles d'initialisation	Nombre de cycles avant d'activer les ports de sortie. Permet d'accumuler des paquets dans les VOQ.
Nombre de paquet moyen par file VOQ	Nombre de paquets (équivalents à des paquets avec MUL_FACTOR_BW_DL à 1) par file moyen attendu avant que les ports de sortie soient activés (initialisation).

Il est à noter que la profondeur des VOQ, bien qu'elle soit en termes de paquets, est en fonction de la longueur moyenne des paquets par entrée (MUL_FACTOR_BW_DL définit au tableau 2.3). Le nombre de cycles et le nombre de paquets moyen par VOQ à l'initialisation peuvent être utilisés indépendamment ou conjointement. Si les deux paramètres sont utilisés, les deux contraintes doivent être satisfaites avant qu'un premier paquet ne puisse sortir du TI. Le nombre de paquets moyen par VOQ à l'initialisation est aussi en termes de paquets avec MUL_FACTOR_BW_DL à 1. Il est donc relatif à la taille moyenne des paquets par port lorsque celle-ci est connue dans le but d'avoir une représentation juste des mémoires. Donc, on simule correctement qu'un paquet dans un VOQ d'une entrée qui possède en moyenne des paquets deux fois plus gros prend deux fois plus de cases mémoire qu'un paquet pour une entrée régulière.

2.1.4 Description modulaire

Chacun des éléments qui constituent l'environnement de validation est présenté dans cette section.

2.1.4.1 Tissu d'interconnexion

Le but du TI dans cet environnement est de permettre une aisance de modélisation de différentes architectures tout en étant précis en termes de cycles. Donc, lorsque possible, une approche à haut niveau d'abstraction est prise. Ainsi, toutes les transactions d'entrée/sortie sont des transmissions de paquets et elles demeurent sous cette forme lors du traitement dans le tissu. Les FIFO qui servent de VOQ et de tampon en sortie traitent des paquets pour simplifier la modélisation. Il en est de même pour les transmissions dans le commutateur crossbar. Ceci est réalisé à l'aide de compteurs internes au TI qui évaluent les temps de transmission. Un paquet passe au travers d'une connexion après un

certain nombre de cycles qui dépend de la taille du paquet et de la largeur du bus sur lequel il est transmis. Pendant cette transmission de paquet, le chemin de données correspondant du commutateur crossbar n'est pas disponible, ce qui a pour effet de bloquer les autres paquets destinés à une même destination ou provenant d'une même source.

Comme il est décrit au chapitre 1, la communication transactionnelle sur les ports se fait à l'aide d'appels de fonctions d'écriture/lecture. La fonction d'écriture calcule la longueur du paquet en additionnant la taille de tous les champs de l'entête avec la taille des données. En connaissant la largeur des ports, le nombre de cycles de transmission est déterminé. Ensuite, il y a une attente correspondant au nombre de cycles calculés. Cette même fonction est responsable d'écrire dans les diverses files VOQ en fonction de la source et de la destination. Le compteur de paquets par file est incrémenté et l'ordonnanceur est avisé par une variable booléenne si un nouvel ordonnancement est nécessaire. Cela se produit lorsqu'une file vide vient de recevoir un paquet. En fonction d'un paramètre de configuration, si la file est pleine, le paquet le plus récent ou le plus vieux est laissé de côté (« packet drop »). La fonction de lecture génère, quant à elle, une information pour chaque port indiquant si la sortie est prête à lire un autre paquet. La fonction est ensuite en attente qu'un paquet soit présent dans la file de sortie concernée. Ensuite, de façon similaire à la fonction d'écriture, le paquet est lu, le temps de transmission est calculé et une attente égale à cette période est faite. La fonction complète ensuite son exécution puisque le paquet est alors complètement reçu.

À chaque cycle, des fonctions internes font évoluer les compteurs de cycles internes et réalisent les lectures/écritures des VOQ/FIFO de sortie lorsqu'elles sont nécessaires tout en respectant les temps de transmission.

Le TI possède trois fils d'exécution (« sc_thread ») principaux. Un « sc_thread » commence l'exécution une seule fois en début de simulation. Chacun contient une boucle

qui s'exécute à chaque cycle. Ceci est fait grâce à une commande d'attente à la fin de la boucle. Cela assure que le fil d'exécution soit exécuté à chaque cycle. Le premier fil d'exécution renferme la fonctionnalité principale du TI alors que le deuxième concerne les files de sortie. Le troisième est simplement formé d'un compteur des paquets écrits dans le TI et sert à l'initialisation. Le premier fil d'exécution est subdivisé en plusieurs fonctions appelées à chaque cycle afin de subdiviser les diverses fonctionnalités et les rendre aisément configurables. Par exemple, pour la première fonction appelée après l'initialisation, soit l'ordonnancement, l'ordonnanceur est sélectionné en fonction de la configuration. Mais, avant la fonction d'ordonnanceur, il y a des états d'attente sur le nombre de cycles et de paquets moyen par VOQ avant que l'ordonnanceur active le TI.

L'ordonnanceur analyse chacune des connexions possibles afin de vérifier si une nouvelle connexion peut être établie dans des circonstances spécifiques. Les trois circonstances qui demandent une nouvelle analyse des connexions par l'ordonnanceur sont :

- Une des VOQ possède une donnée alors qu'elle n'en possédait pas le cycle précédent;
- La transmission dans le TI pour une des connexions vient de se terminer;
- Une file de sortie n'est plus pleine.

La dernière condition ne devrait pas se produire si les files de sortie sont de taille suffisante et que les ports de sortie ne sont jamais bloqués.

Une table à deux dimensions inclut l'information du nombre de paquets planifiés par connexion (i.e. par paire de ports entrée/sortie). Selon l'algorithme de l'ordonnanceur choisi (soit WWFA ou WRR dans notre cas), un certain ordre d'analyse de ces connexions est choisi et un certain mécanisme de rotation de la priorité est établi. Cet ordre d'analyse change après chaque appel de l'ordonnanceur. Pour être assignée, une connexion doit être dans l'une des deux circonstances décrites ci-haut. Cela est détecté

par la vérification de tables de valeurs booléennes qui identifient ces conditions pour chaque connexion. Une fois une nouvelle décision d'ordonnancement prise, le nombre de paquets à transmettre est décidé. Il sera égal à la plus petite valeur entre le nombre de paquets présents dans la file concernée et le nombre de paquets maximal à transmettre par décision. Cette dernière valeur est dictée par l'algorithme d'ordonnancement et la configuration de cet algorithme.

Le commutateur crossbar n'a pas de fonction propre et est abstrait dans le comportement de l'arbitre. L'arbitre doit tenir en compte que le medium de transmission est un commutateur crossbar, donc qu'il ne peut connecter qu'une seule fois chaque entrée vers les ports de sortie et que chaque sortie ne peut recevoir qu'un paquet à la fois.

La fonction qui suit l'ordonnancement applique les connexions déjà décidées pour transmettre les paquets aux files de sortie via le commutateur crossbar conceptuel. Cela a lieu seulement quand le « sc_thread » des files de sortie active l'écriture de paquet. Ensuite, une fonction analyse si l'une des connexions doit être enlevée de la table d'ordonnancement. Cela est fait lorsque le dernier paquet d'une connexion établie précédemment a terminé sa transmission. L'alternance « round-robin » de l'ordre de connexions (selon leur priorité) est ensuite appliquée suivi de l'impression de l'état des files à chaque cycle de façon systématique. Cela permet l'analyse de la base de données créée de façon à recueillir les statistiques désirées.

Le « sc_thread » qui implémente les files de sortie propage initialement les lectures à l'ordonnanceur, calcule les temps de transmission des paquets et contrôle le module d'envoi de paquets qui implémente le commutateur crossbar virtuel en fonction de ce temps de transmission.

Cette façon de faire permet d'avoir une implémentation du TI relativement simplifiée et permet une modification aisée des paramètres d'architecture divers. Ceux-ci

peuvent être l'accélération du commutateur crossbar ou l'ordre d'analyse des connexions dicté par l'arbitre sélectionné. Également, certains ajouts, pouvant potentiellement améliorer la performance, peuvent aisément être implémentés à haut niveau, simplifiant grandement l'expérimentation de ces idées.

2.1.4.2 Modules de test

Les modules de test définissent le trafic d'entrée/sortie sous forme de transactions. La description du comportement par port est ainsi simplifiée. Un module SystemC qui utilise des paramètres provenant d'un fichier de configuration est connecté à chacun des ports. Donc, pour des cas typiques, l'usager n'a qu'à manipuler le fichier de configuration. Ce dernier fichier inclut les définitions des divers paramètres par des instructions de pré-compilation et des variables globales faisant partie d'un « namespace » C++. Pour rendre l'environnement plus simple à utiliser et éviter de recompiler pour changer de configuration, il serait possible d'inclure ces configurations dans un fichier texte qui serait lu à l'exécution pour définir les paramètres configurables. Ceci serait particulièrement pratique si l'on désire simuler un grand nombre de tests, mais cette possibilité a été laissée comme implémentation ultérieure par souci d'économie de temps de développement. Le tableau 2.3 montre la liste des paramètres configurables et leur impact sur le trafic.

Tableau 2.3 Paramètres utilisés par le module de test pour générer le trafic

Nom du paramètre	valeur/ unité	Description
NB_PACKETS	Nb. paquets	Nombre maximum de paquets transmis par port
RANDOM_DEST	on/off	Lorsque désactivé, le port 0 est utilisé comme destination (collisions à la destination)
		Lorsque activé, chaque paquet a une destination aléatoire
BW_MIN_DL	Nb. octets	Taille de base minimum des paquets (pour tous les ports et qui est multiplié par MUL_FACTOR_BW_DL – voir ci-bas)
BW_MAX_DL	Nb. octets	Taille de base maximum des paquets (pour tous les ports)
BW_PERC_BW	0-100 %	Rapport d'utilisation de base en termes de cycles des ports d'entrée
MUL_FACTOR_BW_PERC_BW [NB_PORT]	Ratio	Tableau de facteurs multiplicatifs du rapport d'utilisation de base par port (i.e. multiplie BW_PERC_BW)
MUL_FACTOR_BW_DL [NB_PORT]	Ratio	Tableau de facteurs multiplicatifs des dimensions de base (BW_MIN_DL & BW_MAX_DL) des paquets par port
N_CONSEC	Entier	Le nombre de paquets consécutifs d'une même source forcés à avoir une destination différente
SCV	booléen	Utilise SCV ou non (fonctionnalité de débogage)
NB_PACKET_SAME_DEST	Entier	Nombre de paquets consécutifs en entrée du TI vers une même destination par port (« stream »).

Deux cas typiques de destinations sont aisément configurables : destination aléatoire et une seule destination. Ce dernier cas permet de forcer des collisions au port de destination. Il s'agit de deux scénarios typiquement intéressants. La combinaison de plusieurs paramètres permet d'obtenir aisément différents rapports de trafic et différentes tailles moyennes des paquets par port. Le rapport de trafic de base (BW_PERC_BW) est multiplié par port par un facteur multiplicatif (MUL_FACTOR_BW_PERC_BW) afin d'ajuster le trafic par port aisément. La taille des paquets est, quant à elle, aléatoire dans un intervalle donné par l'usager. Il peut, évidemment, être fixé en donnant des valeurs minimales et maximales de base similaires. Par port, ces valeurs minimales et maximales de base sont multipliées par un facteur (MUL_FACTOR_BW_DL), ce qui facilite la génération de scénarios de trafic divers. Le nombre de paquets successifs générés possédant une destination différente (« N_CONSEC ») est aussi défini. C'est-à-dire qu'on peut décider, par exemple, que trois paquets consécutifs d'un même port source aient trois destinations différentes. Le « N_CONSEC » serait fixé à trois dans un tel scénario.

Pour des besoins spécifiques, l'usager peut également intégrer son propre module de test. Il n'a qu'à créer un module de test systemC et le connecter dans le module « sc_top » pour remplacer les modules génériques utilisés par l'environnement. Donc, malgré que l'environnement soit aisément configurable, il ne sacrifie pas la flexibilité en permettant à l'usager de coder ses propres tests plus complexes. Ceci vient avec le coût plus élevé d'une modification plus complexe.

Pour réaliser la génération aléatoire contrainte, une classe de contrainte qui hérite de la classe « scv_constraint_base » est créée (« packet_base_constraint »). Cette classe utilise les outils fournis par la bibliothèque SCV. Cette classe applique certaines contraintes définies dans le fichier de configuration (RANDOM_DEST, BW_MIN_DL, BW_MAX_DL, N). Elle assigne également d'autres paramètres, telles que la source et la destination, en plus de produire une chaîne de caractères hexadécimale qui forme les données. La fonction « next » de SCV permet d'appliquer les contraintes à la génération

aléatoire d'une nouvelle donnée, soit un objet complexe ou une donnée pointée par un « scv_smart_ptr ». Il s'agit d'une fonction de la classe « scv_constraint_base » qui est surchargée par la classe « packet_base_constraint ». Dans la fonction « next » de cette dernière classe, il y a d'abord un appel à la fonction « next » de SCV pour produire l'entête du paquet avec les contraintes définies et, ensuite, le traitement particulier nécessaire pour l'allocation dynamique avec des générations aléatoires contraintes des caractères hexadécimaux qui forment le champ de données.

L'interfaçage du module de test avec le TI se fait avec diverses fonctions transactionnelles. Les fonctions disponibles sont l'écriture d'un paquet, la lecture d'un paquet et l'attente d'un ou de plusieurs cycles.

Le module de test d'écriture contient un fil d'exécution qui exécute une boucle pour produire un nombre de paquets égal à NB_PACKETS. Il crée d'abord un objet de type « packet_base_constraint » dans le cas où SCV est activé dans le fichier de configuration. En effet, si SCV est désactivé, des paquets sont générés de façon déterministe. Il s'agit en fait d'une fonctionnalité de déverminage qui permet, entre autres, d'utiliser une partie de l'environnement avec SystemC sans SCV. Si SCV est activé, la fonction « next » est appelée pour le paquet et un bit de transmission généré aléatoirement est produit lors de chaque production de paquet. Le bit de transmission est utilisé pour implanter le pourcentage d'utilisation du lien puisque seul le paquet est généré par la classe « packet_base_constraint ». Le bit de transmission permet d'obtenir la distribution spécifiée dans le fichier de configuration. C'est-à-dire qu'il sera à « 1 », par exemple, 70 pourcent du temps pour un trafic spécifié à 70%. Si le bit est à « 1 », le bit est transmis. Sinon, un temps d'attente égal au temps de transmission sur le lien du paquet généré est appliqué par l'appel de la fonction d'attente de haut niveau qui interface avec le TI. Cela permet de bien représenter la bande passante demandée par port. Dans le cas de destinations aléatoires, une analyse est faite à la production de chaque paquet afin de respecter la contrainte de « NB_PACKET_SAME_DEST ». La

destination générée aléatoirement est écrasée par la destination précédente dans le cas où des paquets consécutifs doivent avoir une même destination. Un compteur de paquets sert à produire un ID spécifique par paquet pour une source/destination donnée. Celui-ci est inclus dans le paquet (« *packet_id* »). Le paquet est ensuite transmis au TI via la fonction d'écriture interfaçant au TI. Ce paramètre « NB_PACKET_SAME_DEST » permet de créer des paquets formant une même communication (un même « stream ») vers une destination spécifique.

Le module de test de lecture est simplement constitué d'une boucle qui demande une lecture de paquet via la fonction qui interface au TI à chaque exécution. Comme la fonction de lecture ne retourne que lorsqu'un paquet est prêt, ce module ne contient aucune autre instruction d'attente que la lecture. Le temps de transmission à la sortie est déjà simulé dans le TI.

2.1.4.3 Modules de collecte de données (moniteur)

Ce module transfère simplement les requêtes et les envois de paquets entre le test et le TI. L'impression du temps et du contenu des paquets transférés se fait dans un fichier par ce module. Initialement dans ce projet, cette impression pouvait se faire dans le TI ou dans le module de test selon la configuration sélectionnée par l'utilisateur (et le niveau d'implémentation auquel était simulé le test). La collecte est devenue modulaire puisque cela amène quelques avantages : le même module peut être utilisé pour créer des bases de données de même format lors de toutes les étapes du projet; les nouveaux modules de test spécifiques et nouvelles architectures de TI conçus par l'usager n'ont pas à se soucier de la collecte. Le module peut aussi être utilisé pour les communications avec un TI matériel pour lesquelles il est moins évident de créer une base de données à partir du TI. Le module qui génère la base de données étant le même, cela permet de faire les mêmes analyses statistiques en tout temps sans se soucier du format de la base de données.

Ce module n'étant qu'un intermédiaire, il possède et appelle les mêmes fonctions d'écriture, lecture et attente que le module de test et le TI. Et ce, même si ce module ne joue aucun rôle pour certaines fonctions (comme l'attente, par exemple) puisqu'il n'y a aucune donnée à imprimer.

2.2 Concepts et description de l'environnement de vérification

L'environnement de validation de performance présenté à la section précédente sert de concept de base pour l'élaboration de l'environnement de vérification présenté dans la présente section.

2.2.1 Généralités

L'environnement de vérification utilise les mêmes tests transactionnels que l'environnement de validation. Ceci permet d'obtenir les mêmes scénarios de trafic à fournir au TI matériel lorsque ce dernier est prêt à être testé. Cependant, des modules de conversion de transaction (« transactor ») sont nécessaires pour convertir l'interface transactionnelle en signaux qui s'adressent au TI matériel. La figure 2.2 représente l'environnement de vérification.

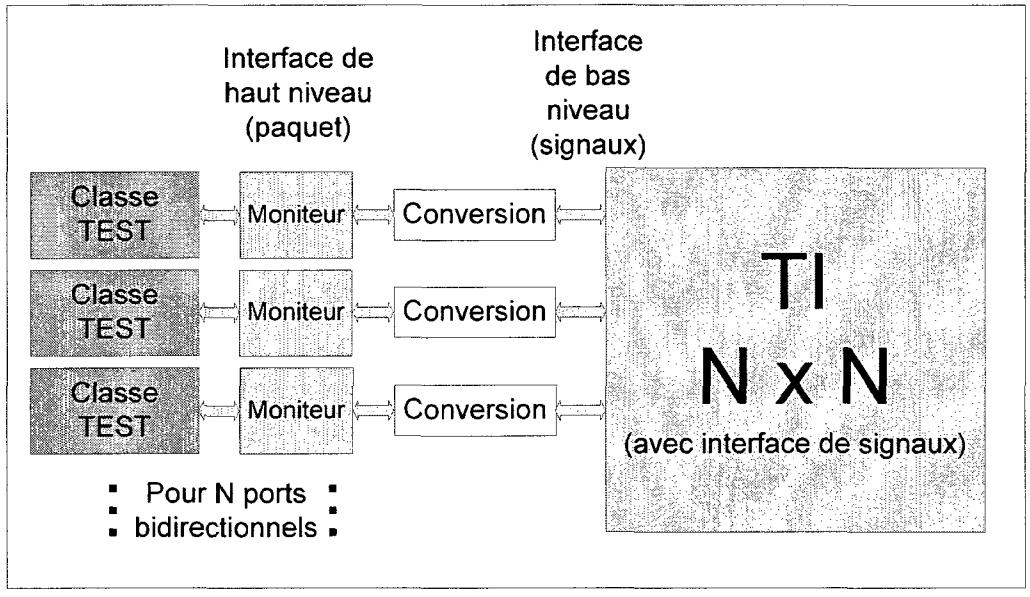

Figure 2.2 Vue d'ensemble de l'environnement de vérification

2.2.2 Description du modèle de paquets

Les paquets sont composés des mêmes informations que celles présentées plus tôt dans ce chapitre. Cependant, pour le modèle de TI matériel, un paquet est divisé en plusieurs mots qui sont transmis en plusieurs cycles sur les différents ports du TI. Les mots d'entrée/sortie sont organisés selon la structure illustrée à la figure 2.3.

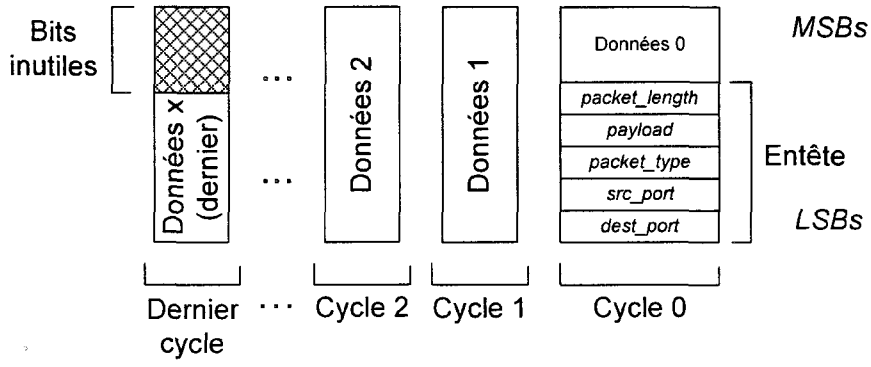

Figure 2.3 Structure d'un paquet divisé en plusieurs cycles

La dimension des différents champs (« *dest_port* », « *src_port* », etc.) sont configurables. Certains paramètres de l'entête, telles que la destination et la source, se doivent d'être dans le premier cycle d'envoi pour que le TI puisse fonctionner efficacement. Cela est un élément intéressant pour notre architecture puisque, par exemple, dès l'arrivée d'un paquet dans le TI, il doit être envoyé au VOQ approprié en fonction de la destination du paquet. On évite ainsi un étage supplémentaire de mémoire tampon aux entrées du TI. À partir des dimensions des différents champs de l'entête et de la dimension des données envoyées, le nombre de cycles du paquet sur un port est déterminé. Le dernier cycle a de bonnes probabilités d'avoir certains bits de données inutilisés, dépendamment de la taille du paquet donnée par « packet_length ».

2.2.3 Description modulaire

Les modules de l'environnement de vérification qui diffèrent de ceux utilisés pour l'environnement de validation (expliqués antérieurement) sont présentés ici.

2.2.3.1 Tissu d'interconnexion

Le tissu d'interconnexion matériel à nombre de ports générique est réalisé en VHDL. Il possède quatre principales fonctionnalités. La première permet d'effectuer des lectures/écritures des données d'entrée dans des FIFO servant à mémoriser les données. La seconde fonctionnalité consiste à prendre une décision sur le routage des paquets. La troisième permet de gérer l'envoi des données sauvegardées vers un des ports de sortie en fonction de l'adresse de destination et de la décision de l'ordonnanceur. La dernière partie est contrôlée par le module de gestion de l'envoi des paquets et fait simplement router les paquets vers les bonnes sorties selon les instructions reçues. Dans le cadre de ce projet, cette dernière partie est un module communément appelé commutateur crossbar. Le schéma illustrant les divers modules du tissu matériel est montré à la figure 2.4.

Le rôle principal du module VOQ est de gérer les écritures et lectures des divers FIFO d'entrée qu'il contient. Ces FIFO servent de files de sortie virtuelles. Il y a un module VOQ pour chacun des ports d'entrée qui contient autant de FIFO que le TI possède de ports de sortie. Chaque paquet d'une entrée est dirigé vers le FIFO correspondant à sa destination. Le même comportement est appliqué pour chaque module VOQ correspondant aux autres entrées. Ainsi, on s'assure que les entrées soient traitées de façon indépendante. Cela permet aussi de bien ordonner les paquets pour mieux planifier leur transfert par la suite. On a de cette manière une bonne connaissance des paquets contenus dans le TI en termes de source/destination, ce qui nous permet de router efficacement les paquets.

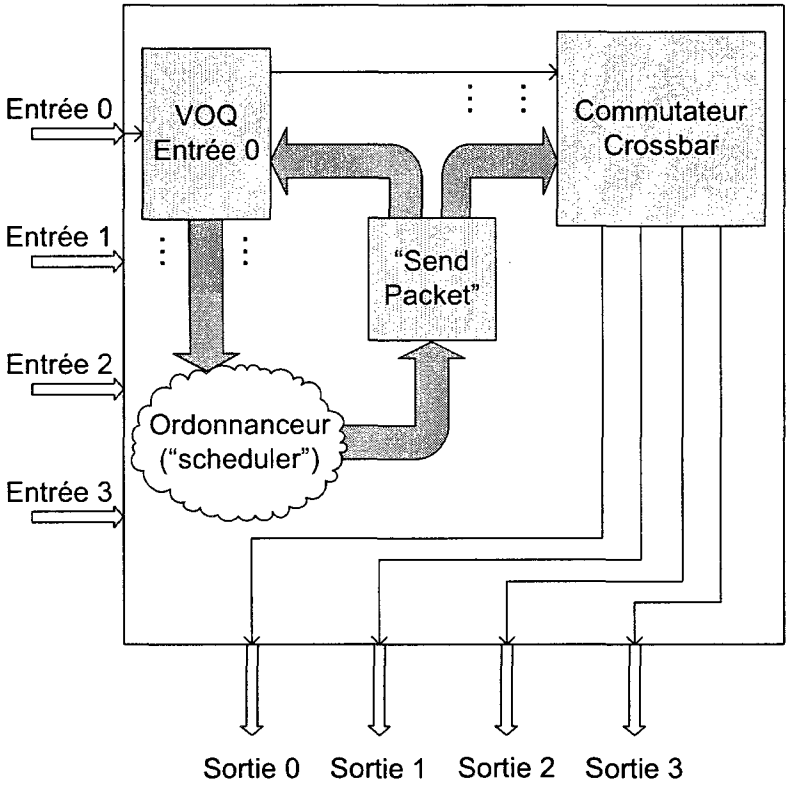

Figure 2.4 Schéma bloc du tissu d'interconnexion matériel

Le module ordonnanceur est en quelque sorte un arbitre et il doit faire une planification dans trois situations particulières. En effet, lorsque la planification est faite, une décision est prise pour le routage des paquets. La prochaine planification aura lieu lorsqu'un évènement se produira. Un des évènements qui enclenche ce processus est l'arrivée d'un paquet complet dans une file de VOQ alors que celle-ci était précédemment vide. Dans ce cas, le module VOQ est en charge d'aviser l'ordonnanceur. Le deuxième évènement possible est lorsqu'une sortie précédemment occupée se libère et est maintenant disponible pour l'envoi de nouveaux paquets. Dans une telle situation, le module de gestion de l'envoi de paquets doit aviser l'ordonnanceur en lui spécifiant quelles sorties sont maintenant prêtes à recevoir. Enfin, le dernier évènement possible est lorsqu'un port de lecture envoie une commande de lecture alors qu'il n'était pas en mode

de lecture au cycle précédent. Typiquement, cette dernière condition est moins fréquente car, en temps normal, on peut supposer que la destination est toujours prête à recevoir des paquets. Dans une situation concrète pour laquelle on veut simuler un cas de test particulier, il est pertinent d'attendre quelques cycles en début de simulation afin de laisser les éléments de mémoire du TI se remplir. Ce dernier évènement devient alors essentiel.

Le module d'envoi de paquets accepte la nouvelle planification de l'ordonnanceur et adapte son information locale d'après le nouveau plan. Évidemment, ce nouveau plan n'est effectif que pour certaines connexions spécifiées par l'ordonnanceur, car ce n'est pas nécessairement tous les ports qui ont eu besoin d'une nouvelle planification. Ce module gère l'envoi de paquets en plusieurs cycles ainsi que l'envoi de plusieurs paquets déjà autorisés par l'ordonnanceur. Lorsque l'envoi est terminé, ce module fait une requête à l'ordonnanceur pour obtenir une nouvelle planification pour certains ports spécifiés dans la requête.

Le commutateur crossbar, quant à lui, exécute le transfert des paquets selon les commandes reçues du module d'envoi de paquets et les données reçues du module VOQ. Il existe plusieurs façons de réaliser cette connexion. Une architecture de type commutateur crossbar est implantée avec des multiplexeurs qui dirigent une entrée spécifique (un des FIFO de un des modules VOQ en entrée) vers une seule sortie à la fois. L'arrangement exact de multiplexeurs et fils d'un commutateur crossbar est présenté à la figure 1.4 du chapitre 1. Cela se distingue par exemple d'une architecture « fully connected » [39] qui permettrait à une seule entrée d'envoyer vers plusieurs sorties à la fois (i.e., avec une architecture « fully connected », une même source peut écrire vers plusieurs destinations à la fois des paquets différents provenant de FIFO différents pour cette même entrée). Dans le cas de l'architecture « fully connected », il y a un gros multiplexeur par port de sortie qui doit permettre à tous les VOQ de communiquer avec chaque sortie, ce qui fait que le nombre de chemins de données (qui doivent parcourir de

grandes distances) est considérablement plus élevé. Le nombre de chemins de données croît de façon quadratique en fonction du nombre de ports [39]. De plus, des structures matérielles nettement plus complexes doivent être implantées aux sorties pour stocker plusieurs paquets qui seraient transmis simultanément. Ceci se distingue grandement du commutateur crossbar qui possède un seul chemin par entrée et un seul chemin par sortie et qui n'a pas besoin de gérer la réception de paquets simultanés aux sorties. On voit bien l'impact sur la surface de silicium et sur l'architecture d'utiliser un commutateur crossbar au lieu d'un réseau « fully connected ».

Les tableaux 2.4 et 2.5 décrivent respectivement les paramètres génériques du module TI et ses ports. Tous les paramètres des ports du TI étant génériques (tableau 2.4) et puisqu'un TI a une interface simple, les ports du tableau 2.5 sont suffisant pour modéliser les ports d'interface de n'importe quel TI avec des communications d'entrée qui ne peuvent être interrompues, ce qui est généralement le cas.

Tableau 2.4 Liste des paramètres génériques du TI

Générique	Type	Description
FIFO_DEPTH	Entier	Profondeur du FIFO (en nombre de mots)
NB_PORT	Entier	Nombre de ports du TI
NB_PORT_LOG2	Entier	Log_2 du nombre de ports (arrondi vers le haut)
NB_BITS	Entier	Taille du mot d'entrée / sortie
NB_BITS_ADDR	Entier	Taille des adresses de source / destination contenus dans le paquet
NB_BITS_PACKET_ID	Entier	Taille d'un champ définissant l'identificateur de paquet (« id »)
NB_BITS_PAYL	Entier	Taille d'un champ définissant le « payload » du paquet (pas utilisé pour ce projet)
NB_BITS_DATA_L	Entier	Taille d'un champ définissant la taille des données contenues dans le paquet en octet et en n'incluant pas la taille de l'entête du paquet

Tableau 2.5 Description des ports du TI matériel

Champ	Direction	Taille (bits)	Description
sys_clk	IN	1	L'horloge
sys_reset_n	IN	1	Reset asynchrone actif bas
wr	IN	NB_PORT	Il précise que le port courant est prêt à écrire dans le FIFO correspondant
in_port	IN	NB_PORTs mots de NB_BITS	Les mots à l'entrée
read	IN	NB_PORT	Il précise que le port de sortie courant est prêt à lire un paquet
out_port	OUT	NB_PORT PORTs mots de NB_BITS	Présente la sortie du TI (retirée du FIFO et envoyée vers la sortie correspondante)
out_valid	OUT	NB_PORT	Indique que la sortie correspondante envoie une donnée valide en sortie.

Le TI matériel n'étant pas l'objet principal de ce mémoire, les aspects d'implémentation plus détaillés de celui-ci sont donnés en annexe A. Le but de ce module, dans le contexte du projet, est principalement de prouver la connectivité d'un TI matériel dans l'environnement créé pour former un banc d'essai pour la vérification. Pour ce faire, un module matériel fonctionnel devait être construit.

2.2.3.2 Module de conversion de transactions (« Transactor »)

Le module de conversion de transactions (« transactor ») est un module qui permet de faire le pont entre deux niveaux d'abstraction d'une interface. L'interface à haut niveau d'un module SystemC est accessible via des appels de fonction. Dans le cadre de ce projet, les fonctions de lecture, d'écriture et d'attente sont celles communément utilisées. Afin d'avoir des modules de test réutilisables, les fonctions d'interface en termes de transactions sont les mêmes que celles du TI à haut niveau. Donc, pour les modules de test, cet accès au module de transaction n'a pas à être connu.

Le retour de l'appel de fonction est fait quand la transaction est complétée. Il s'agit d'un des deux aspects de la conversion.

Le deuxième aspect est l'interface matérielle. Celle-ci se connecte au TI pour ce projet. Il s'agit d'une interface de signaux physiques, telle que discutée à la section précédente. Lors d'un appel de la fonction d'écriture par le module de test, ce module active un port d'écriture et fait suivre les données jusqu'à ce que le paquet soit complètement transmis. Le paquet qui est inclus dans la transaction est en fait segmenté en fonction de la taille du port physique. Une fois le paquet transmis au complet, le contrôle est retourné au module de test. Lors d'un appel de lecture, le module permet la réception de mots du TI. Il est ensuite en attente ou il accumule un paquet complet provenant du TI. L'entête du paquet permet de déterminer la taille et donc la fin du paquet. Ensuite, le contrôle est redonné au module de test avec le paquet reçu. Cela peut prendre un nombre important de cycles en cas de trafic léger. La fonction d'attente, quant à elle, ne fait aucune lecture ou écriture. Elle bloque l'accès au port pour un certain nombre de cycles défini dans l'appel de la fonction en appelant la fonction d'attente vers le TI.

Les ports de l'interface physique de ce module sont présentés au tableau 2.6. Il est à noter que le module de conversion a été construit avec une interface différente du TI matériel présenté et légèrement plus générique pour permettre plus de flexibilité selon le TI matériel utilisé. L'interface permet au port d'entrée de bloquer les écritures au lieu qu'un paquet soit perdu dans le TI, comme il est fait dans notre implémentation matérielle. Les connexions simples à réaliser au TI matériel de la section précédente sont également montrées au tableau 2.6.

Tableau 2.6 Description des ports matériels du module de conversion

Champ	Direction	Taille (bits)	Description	Connexion au TI
sys_clk	IN	1	L'horloge	sys_clk
rst_n	IN	1	Reset asynchrone (actif bas)	sys_reset_n
write	OUT	NB_PORT	Il précise que le port courant est prêt à écrire au TI (« 1 »)	wr
data_write	OUT	NB_PORTs mots de NB_BITS	Les mots du paquet à écrire au TI	in_port
port_full	IN	NB_PORT	Indique que le port d'écriture de paquet correspondant vers le TI est plein (bloquée, « 1 »)	'0' (port non bloqué)
read	OUT	NB_PORT	Il précise que le port de lecture courant est prêt à lire un paquet (« 1 »)	read
data_read	IN	NB_PORT PORTs mots de NB_BITS	Présente les mots du paquet lu du TI	out_port
port_empty	IN	NB_PORT	Indique que le port de lecture est vide et donc, qu'on ne reçoit pas de mot d'un paquet valide (« 1 »).	not(out_valid)

Le module de transaction permet donc l'interface vers un autre niveau d'abstraction. Il s'intègre aisément à l'environnement à haut niveau et permet la création d'un véritable banc d'essai. Il permet également de conserver un niveau d'abstraction élevé et d'abstraire les détails de communication lors de la création ou de la configuration d'un module de test. Tous les scénarios testés sur une architecture à haut niveau peuvent donc être reproduits pour un TI matériel sans effort additionnel, une fois la conversion réalisée.

2.2.4 Fonctions de vérification utiles

L'environnement étant construit sous forme de classes C++, on a avantage à tirer profit des nombreuses possibilités que ce langage amène. Entre autres, il est fréquent en vérification de vouloir comparer des valeurs, vérifier des propriétés et d'afficher un message ou une erreur allant même jusqu'à arrêter la simulation dans certains cas. Le langage utilisé rend aisé la création de telles fonctions génériques réutilisées à plusieurs reprises dans l'environnement créé. L'effort de vérification est ainsi concentré et simplifié. Dans ce projet, une telle fonction générique de vérification de propriété qui affiche un message d'erreur en cas de propriétés non-respectées a été créé et utilisé à maintes reprises. Elle est utile lors de l'implémentation de cas de tests spécifiques et d'idées architecturales, ainsi que lors de comparaison des résultats avec un modèle. Une bibliothèque de ces fonctions utiles amène un aspect intéressant à l'environnement.

2.3 Dualité de l'environnement de vérification et validation de performance et simulation

En combinant les modules présentés aux sections précédentes, on obtient une plateforme unifiée complète de développement matériel. Tous les éléments sont compatibles. Ceci permet d'avoir un seul environnement pour l'exploration architecturale, la validation de performance et la vérification. Cet environnement permet ainsi d'accroître la productivité d'une équipe de conception matérielle. La figure 2.5 représente cet environnement unifié.

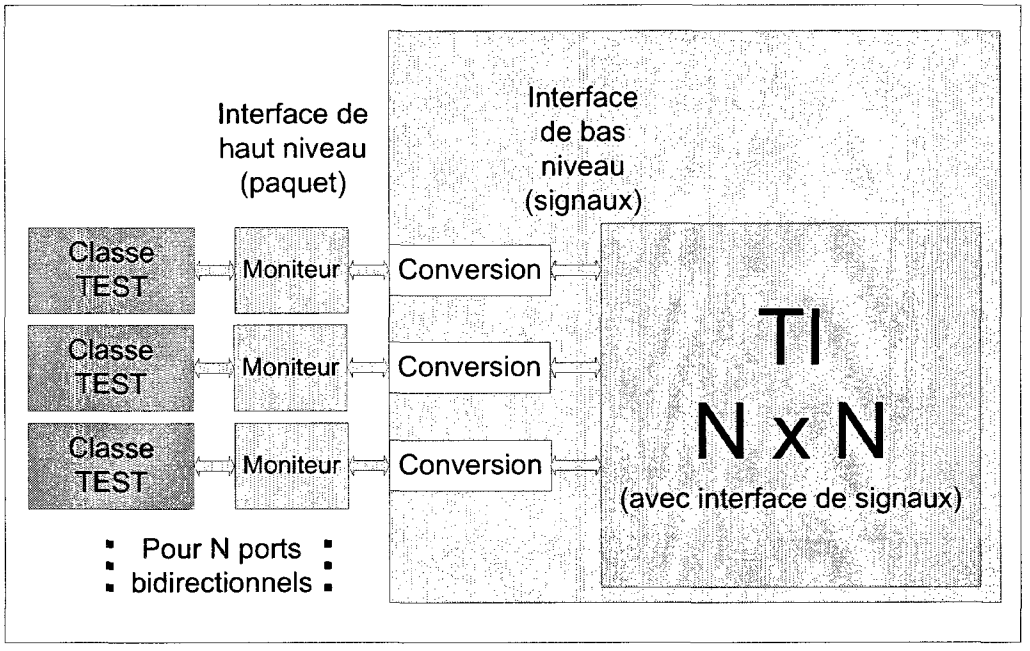

Figure 2.5 Vue d'ensemble de l'environnement de vérification et validation intégré

Tel que mentionné au chapitre précédent, un simulateur multi-langage, comme celui utilisé pour ce projet (Modelsim), donne certaines contraintes supplémentaires par rapport au SystemC de OSCI. On doit en fait créer un « sc_module » qui inclut la déclaration de tous les autres « sc_modules » au plus haut de la hiérarchie. On n'a qu'à simuler ce module tout comme on simulerait un banc d'essai VHDL avec l'interface de Modelsim. Ce « sc_module » (« sc_top ») remplace en quelque sorte le « sc_main » d'un code écrit pour être exécuté dans un environnement OSCI. Donc, ce module « sc_top » inclut la déclaration et la connexion du TI, des modules de conversion de transaction ainsi que des modules de tests. Modelsim requiert que ces modules soient instanciés dynamiquement, ce qui fonctionne aussi en mode OSCI. Dans ce « sc_top », il y a une condition configurable par l'usager pour utiliser le TI matériel ou celui écrit en SystemC, ce qui permet de sélectionner entre la validation à haut niveau et la vérification d'un TI.

Lors d'une simulation avec tissu SystemC pour la validation de performance, un simulateur HDL n'est pas nécessaire. D'ailleurs, théoriquement, l'exécution de code compilé est plus rapide. Afin de permettre une plus grande flexibilité, l'environnement rend possible la simulation OSCI et celle avec un simulateur HDL (Modelsim). Pour ce faire, un « sc_main », qui instancie ce même module « sc_top », a été créé pour la simulation OSCI. Le paramètre « MTI_INTEGRATION » de l'environnement Modelsim est utilisé pour activer ce « sc_main » ou non. S'il existe, la simulation est dans un environnement Modelsim. Le simulateur multi-langage peut donc est utilisé seulement lorsqu'il est nécessaire, soit lorsqu'on vérifie un modèle matériel.

Lors d'une simulation OSCI, divers paramètres, tels que le temps par défaut et le temps total à simuler, sont définis dans le « sc_main ». Dans le cas d'une simulation multi-langages, ces paramètres sont configurés en utilisant l'interface de Modelsim ou à l'aide de scripts d'exécution écrits pour Modelsim.

2.4 Collecte de données et calculs de statistiques

Les techniques pour créer les bases de données et générer les statistiques pertinentes à la validation de performance sont présentées dans cette section.

2.4.1 Collecte de données

Tel que discuté au chapitre 1, SCV inclut une fonctionnalité permettant de collecter des données de manière standardisée. Le principe est intéressant, mais plutôt limitatif. Les bases de données se créent automatiquement selon un format précis et plutôt difficile à interpréter si des informations de plusieurs types sont présentes. Dans notre cas, il y a un bon nombre d'informations différentes : le taux d'utilisation des files à chaque cycle, le temps d'arrivée et d'envoi d'un paquet à un temps spécifique, si un paquet est laissé de côté à un temps spécifique, etc. Pour cette raison, l'utilisation de la

collecte de données de SCV a été rejetée. La collecte de données se fait manuellement avec un module de collecte de données (« moniteur ») par port qui écrit dans un fichier lors de la simulation. Une fois la simulation terminée, ce fichier est parcouru par un script Perl qui analyse et regroupe les données. Les statistiques pertinentes sont ensuite calculées et imprimées à l'écran ou sauvegardées dans un nouveau fichier. L'annexe B montre un exemple du format des données recueillies lors d'une simulation. Les lignes du fichier avec la mention FIFO représentent l'occupation des diverses files à un temps spécifique pour chacun des FIFO. Dans le cas des VOQ, pour un même temps de simulation, chaque ligne montre l'état des VOQ pour une entrée. Pour chaque ligne montrant l'état des FIFO, le nombre de paquets en mémoire est présenté en ordre croissant par port de sortie.

L'état d'utilisation des diverses files (en pourcentage) est imprimé à partir du module qui implémente le TI. Ce paramètre est essentiel à l'analyse complète d'une simulation. Lorsqu'il s'agit d'une simulation avec le modèle de haut niveau, il est aisé d'inclure ces impressions dans le module. Par contre, lorsqu'il s'agit d'un modèle matériel, ce dernier peut imprimer cette information à l'aide d'une partie de code non-synthétisable qui utilise les fonctions d'impression à l'écran de la bibliothèque VHDL standardisée « textio ». Ceci implique une légère adaptation du code matériel et rend l'analyse à partir d'une simulation qui s'exécute sur un émulateur matériel impossible. Par contre, dans un tel scénario, une alternative possible serait de créer un module externe non-synthétisable et simulé à l'extérieur de l'émulateur, mais qui reçoit l'état des différentes files via des ports physiques. Il s'agit là des seules adaptations requises pour le modèle matériel afin de l'intégrer à l'environnement d'émulation et de permettre une analyse complète.

Avant de commencer la simulation, il y a une étape d'impression des divers paramètres de configuration dans la base de données. Il s'agit d'un appel de fonction effectué lors de la création de l'objet « sc_top » qui hérite de « sc_module ». Cet objet

englobe tous les modules du système créé. Il connait la configuration globale et est donc un bon candidat pour imprimer ces paramètres de configuration. Une des raisons de cette impression est d'assurer qu'on connait les divers paramètres lors d'une analyse ultérieure de la base de données créée. Le but principal est cependant de passer l'information au script d'analyse Perl sans avoir à passer des paramètres manuellement au script. Il y a trois types de données de configuration imprimés : configuration d'architecture, configuration des paquets et configuration du trafic (utile en cas de trafic aléatoire contraint). La configuration d'architecture du TI contient les paramètres du tableau 2.2. La configuration du modèle de paquet contient la taille des divers champs de l'entête décrite au tableau 2.1. Enfin, pour ce qui est de la configuration du trafic, elle contient tous les éléments du tableau 2.3, à l'exception du nombre de paquets. Le format d'impression de ces données de configuration est montré à l'annexe B.

2.4.2 Calculs statistiques

Le tableau 2.7 présente les statistiques calculées par le principal script Perl créé et utilisé pour ce projet. L'équité de connexion est mesurée par le rapport des nombres de cycles d'envoi de paquets alors qu'il y a au moins un paquet à envoyer. Ce dernier paramètre a déjà été décrit au chapitre précédent. Il est aisé d'adapter ce script ou d'en créer de nouveaux pour générer d'autres statistiques selon les besoins de l'usager.

Tableau 2.7 Paramètres calculés par le script d'analyse statistique

Paramètre calculé	Par entrée	Par sortie	Par connexion	Total	Jain entrées
Nombre de paquets perdus	Oui	Oui	Non	Oui	Oui
Nombre de paquets envoyés	Non	Non	Non	Oui	Non
Nombre de paquets reçus	Oui	Non	Non	Oui	Non
Équité de connexion	Oui	Non	Oui	Non	Oui
Pourcentage moyen d'utilisation par VOQ	Oui	Oui	Oui	Oui	Oui
Latence moyenne des paquets	Oui	Oui	Oui	Oui	Oui
99^{ième} percentile de latence	Oui	Non	Non	Oui	Oui
Nombre de cycles actifs analysés	Oui	Oui	Oui	Oui	Non
Taux d'utilisation du lien	Non	Oui	Non	Oui	Non

Le script possède trois paramètres configurables indépendants; c'est-à-dire des paramètres programmés qui n'ont aucune incidence sur la simulation et qui sont donc directement dans le script Perl. Il y a d'abord le nombre de cycles avant que les statistiques ne soient prises en compte (*$NB_INIT_CYCLES*). Ce paramètre se distingue du paramètre d'initialisation utilisé par le TI en ce sens qu'il permet simplement d'ajuster le temps de début de la fenêtre temporelle d'analyse pour une simulation. Un autre paramètre qui permet de mettre fin à cette fenêtre d'analyse est le nombre de cycles analysés. Il peut être exprimé en cycles par port (en moyenne) ou en cycle total (*$NB_TOT_DATACYCL_ANAL*). Enfin, le dernier paramètre est le nombre de cycles inactifs avant qu'on considère une simulation terminée et qu'on arrête l'analyse (si cela se produit avant que le nombre de cycles à analyser soit atteint). Il s'agit de *$NB_CYCLE_MAX_IDLE*.

Le calcul statistique se fait par un script Perl. D'abord, le script lit les premières lignes du fichier de base de données afin d'obtenir différents paramètres de configuration.

Une fois cette configuration obtenue, chaque ligne suivante est analysée par le script. Lors d'une lecture dans la base de données de l'état des files VOQ, une analyse est faite à savoir si le temps d'initialisation est passé. Si c'est le cas, le compteur de paquets par connexion qui sont prêts à être transmis est incrémenté pour chaque file possédant des données. Également, l'état de chaque file est ajouté à un compteur pour pouvoir calculer la moyenne d'utilisation des files après l'analyse. Lorsqu'on utilise l'environnement à haut niveau, les files sont en termes de paquets et non de mots. Dans ce cas, la taille moyenne des paquets par port (MUL_FACTOR_BW_DL du tableau 2.3) est considérée dans la statistique calculée. Il est à noter que les lignes d'impression de l'état des VOQ sont utilisées par le script pour faire le suivi du nombre de cycles de simulation puisque celles-ci sont présentes dans le fichier de base de données à chaque cycle de simulation.

Lorsque le script identifie qu'un paquet est écrit vers le TI, le temps est noté dans une table qui inclut chacun des paquets pour toutes les connexions d'entrée/sortie (table 3D). Il y a un identificateur (ID) spécifique par paquet pour chacune des connexions possibles qui permet d'identifier clairement chaque paquet. Le paquet est marqué, par défaut, comme étant perdu jusqu'à ce qu'il soit sorti du TI. S'il ne sort jamais, il demeure identifié ainsi.

Lorsqu'un paquet est lu d'un port du TI, le temps de lecture est noté dans une table 3D similaire à celle pour le temps d'écriture. La latence est calculée pour le paquet et écrite dans une troisième table 3D. La latence moyenne est mise à jour. Le paquet est identifié comme n'étant pas perdu et le temps d'utilisation par connexion est mis à jour. Si l'élément identifié est un paquet perdu, les compteurs de paquets perdus sont mis à jour.

Ensuite, si un nombre de cycles fixe et connu du script se passe sans qu'aucun paquet ne soit reçu aux divers ports, la simulation se termine et les cycles d'inactivité sont enlevés de l'analyse statistique. Cela permet d'éviter de biaiser les statistiques.

À ce point, toute la base de données a été analysée, mais il reste quelques calculs statistiques à faire. Les latences de paquets sont mises en ordre pour calculer le 99^{ième} percentile. Le nombre de paquets perdus est calculé par entrée et par sortie. Le taux d'envoi de paquets alors que le VOQ n'est pas vide, le taux d'utilisation des VOQ, les latences moyennes par port et totales peuvent ensuite être calculés avec les paramètres recueillis. Aussi, en utilisant le nombre de cycles de simulation calculés et le nombre de cycles des connexions utilisées, le script peut identifier le taux de transmission de données aux sorties en pourcentage; 100% indiquant qu'un port est utilisé en tout temps. Le paramètre de Jain est enfin calculé pour tous les paramètres décrits. Lorsqu'un trafic asymétrique est utilisé, un poids sert lors du calcul du paramètre de Jain pour une statistique relative à la taille du paquet par port : le nombre de paquets perdus. Ceci permet de considérer la taille moyenne des paquets perdus des différents ports. Pour tous les autres paramètres de Jain calculés, un poids de 1 est utilisé pour tous les ports comparés et ce, même pour l'utilisation des files VOQ. Cette statistique est en pourcentage et les diverses VOQ, bien qu'instanciées avec une taille en termes de paquets, ont une taille normalisée en fonction du trafic moyen. C'est-à-dire qu'un port qui traite des paquets de taille moyenne 5 fois plus grande possède des files qui contiennent 5 fois moins de paquets, mais des paquets de taille plus importante.

Il est à noter que le script est configurable pour analyser des bases de données OSCI ou Modelsim. En effet, Modelsim inclut un caractère « # » au début de chaque ligne lors de l'impression. Il faut que le script en tienne compte. Le script est en mesure d'identifier ce caractère par lui-même.

Il est également possible, avec le script, de produire des données à tracer dans un graphique avec « gnuplot » [14] ou un autre logiciel similaire. Un des cas qui a été implémenté dans le script réalisé est l'évolution de la latence dans le temps. On peut ainsi aisément observer l'évolution de la latence.

2.5 Conclusion

Ce chapitre décrit la réalisation d'un environnement de validation de performance et de vérification à haut niveau d'abstraction qui tire profit d'un langage de programmation pour la vérification avancée. Celui-ci s'intègre de façon efficace à la méthode de conception/vérification industrielle et a pour but de simplifier et optimiser la validation de performance. La génération de paquets demande un effort minimal et se fait en utilisant la génération aléatoire contrainte. Le chapitre suivant présente des compléments à l'environnement développé en y intégrant des modèles de trafic plus complexes.

CHAPITRE 3. INTÉGRATION DE MODÈLES DE TRAFIC COMPLÉMENTAIRES

Le chapitre précédent montre les éléments de base de l'environnement réalisé. Dans le but de permettre une validation et une vérification plus complète, ce chapitre présente des modèles de trafic qui s'intègrent à l'environnement créé.

3.1 Intégration d'une application complète matériel-logiciel

Un des types d'environnement qui permet de faire une validation plus complète est d'avoir, en complément à un environnement de simulation facilement configurable tel que présenté au chapitre 2, une application concrète et réaliste. Une telle application qui s'intègre aisément à notre environnement est présentée dans cette section.

3.1.1 Concepts et description de l'environnement avec XTSC

Les aspects descriptifs de l'environnement qui permet d'avoir une application matériel-logiciel sont présentés ici.

3.1.1.1 Généralités

Dans le cadre du projet, une application avec processeur qui interface avec un TI, qui, lui, implémente différents concepts architecturaux, a été réalisée. Les tissus d'interconnexion offrent un grand potentiel aux systèmes embarqués. Dans un tel contexte, l'exemple concret d'un ASIP qui calcule un algorithme particulier de

compensation de mouvement en accédant à un ensemble de mémoires fait le sujet de ces travaux.

Ce modèle permet d'avoir un trafic réel et réaliste parce qu'il s'agit en fait d'une application réelle. Tensilica Xtensa SystemC (XTSC) permet l'instanciation de processeurs configurables dans un environnement SystemC [35]. Puisque XTSC permet d'exécuter n'importe quelle application dans un processeur Tensilica et d'interfacer avec un module externe SystemC quelconque par le PIM (« Platform Independent Model ») de Tensilica, XTSC fournit une plateforme pour implémenter une application réelle qui est compatible avec l'environnement présenté. Puisque XTSC est relativement simple d'utilisation, il est pertinent de l'utiliser pour valider une application réelle rapidement.

Cet aspect du projet a été réalisé en collaboration avec un autre étudiant de l'école Polytechnique travaillant sur la compensation de mouvement [5]. La réalisation intègre des parties tirées de nos projets respectifs. La compensation de mouvement est l'évaluation du mouvement d'une certaine partie de l'image entre deux images complètes d'une séquence vidéo [38]. En l'occurrence, l'application utilisée sert à modifier le taux de trame (nombre d'images par seconde) en insérant des images calculées par un algorithme de compensation de mouvement (MC-FRC) [10]. Dans le but d'accélérer et paralléliser les accès en mémoire, qui peuvent devenir un goulot d'étranglement pour du traitement vidéo, un TI peut être utilisé pour ce type d'application. La figure 3.1 montre l'architecture réalisée.

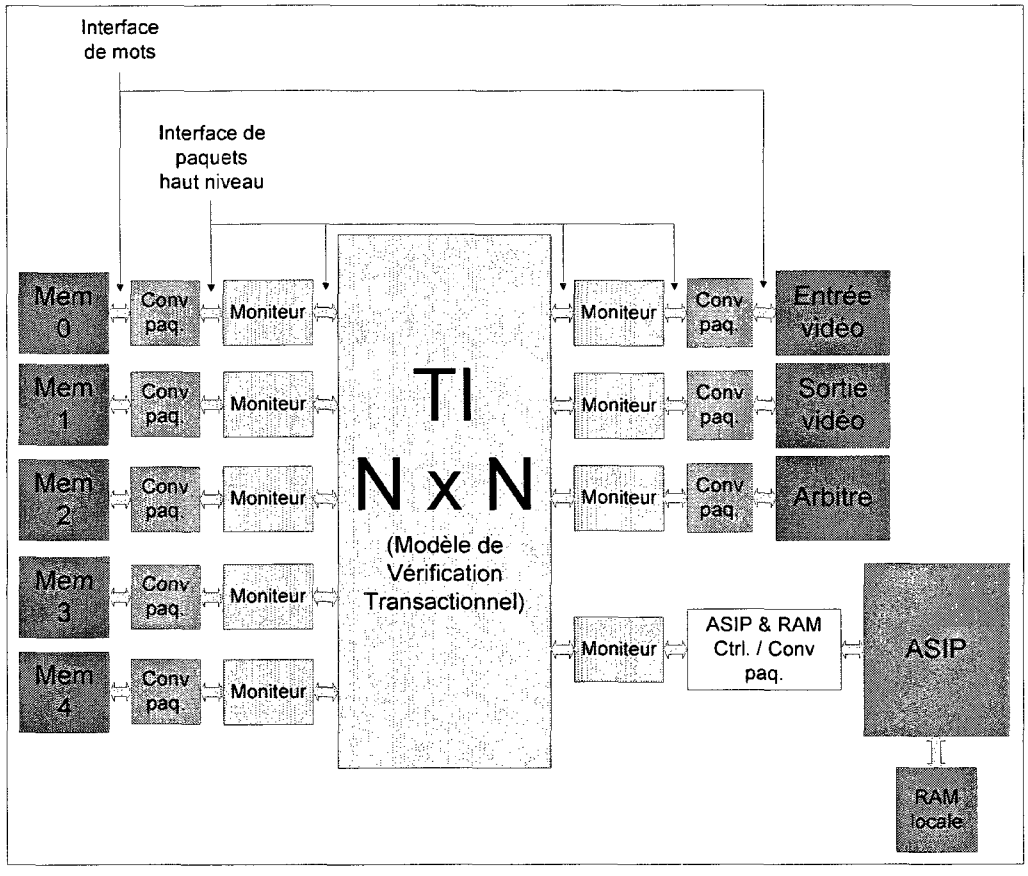

Figure 3.1 Architecture qui utilise un TI pour un calcul de compensation de mouvement fait avec un ASIP

XTSC de Tensilica permet l'instanciation de processeurs, mémoires et autres modules SystemC en créant de simples objets C++. Une structure de paquet est nécessaire pour le transfert des données et des informations de contrôle dans le modèle du TI. Donc, la conversion des paquets (intégrée aux modèles des différents modules) a été créée pour interfacer les modèles des modules réalisés avec le TI à haut niveau présenté au chapitre 2. Les modèles créés sont sous forme de « sc_module ». XTSC montre un exemple d'intégration SystemC avec du code C, tel que cela sera discuté à la section suivante (3.2). Le processeur est encapsulé dans un « sc_module » pour utiliser la

fonctionnalité de plusieurs « threads » virtuels donnée par SystemC. Puisque le processeur est un module complètement indépendant qui utilise une base de temps indépendante du reste des modules SystemC, la synchronisation des modules externes avec le processeur est faite avec une fonction qui retourne le contrôle similaire à celle décrite à la section 3.2. Cette dernière fonction est disponible par un module de conversion d'interface non-bloquant (*if. non-bloquante*) connecté au TI qui appelle les fonctions de lecture/écriture bloquantes vers le TI.

Par souci de rapidité d'accès aux données par l'ASIP (qui, par le fait même, augmente la rapidité d'exécution), différentes mémoires sont utilisées pour les images successives. Pour un calcul de compensation de mouvement, deux images sont nécessaires pour le calcul de l'image courante : la suivante et la précédente. Les deux images sont sauvegardées dans des mémoires externes par le module d'entrée vidéo. Pendant ce temps, l'ASIP déplace les images à partir des mémoires externes vers les mémoires locales, calcule la compensation de mouvement et envoie le résultat vers les mémoires externes à nouveau. L'arbitre connecté à un port du TI gère les divers transferts entre les différents modules. Les requêtes de transfert des images sont envoyées au module d'entrée vidéo par l'arbitre, qui agit comme maître. Ce même arbitre génère un paquet indiquant au module de sortie vidéo à quel moment commencer à transmettre l'image calculée ou l'image source vers la sortie du système. De même, après avoir reçu un paquet indiquant de débuter l'opération de la part de l'arbitre, le module de contrôle de l'ASIP (*ASIP & RAM Ctrl.*) commence ses transferts de données vers/de ses mémoires locales. Une fois les données sources copiées en mémoire locale, l'ASIP entreprend ses calculs. Ce processus a besoin d'être fait en temps réel pour une application vidéo.

L'architecture peut être étendue à un traitement à plusieurs processeurs afin de prendre avantage des bienfaits d'un TI. C'est-à-dire que cela permettrait de prendre avantage de l'aisance de traiter un grand nombre de communications simultanées avec un TI. L'ajout de processeurs supplémentaires peut permettre, par exemple, d'atteindre un

calcul en temps réel pour une image de taille plus importante. Pour ce faire, l'image peut être subdivisée pour être traitée par plusieurs processeurs, montrant l'efficacité de l'architecture de base choisie. Ceci mènerait à un trafic plus complet et réaliste dans le TI et pourrait faire l'objet d'un travail ultérieur.

3.1.1.2 Modèle de paquet

Le modèle de paquet utilisé est, à une seule exception près, le même que celui présenté au chapitre 2. Le champ de données a été modifié pour être une structure qui inclut, en plus des données, une seconde couche de mots de contrôle. Cela inclut l'opération à effectuer, un champ d'adresse qui inclut un ou deux port(s) du TI pour l'opération effectuée et une longueur du champ de données à transmettre. L'opération prend un octet. Le champ d'adresse (port du TI) et le champ indiquant la longueur des données (en termes d'octets) ont tous les deux une taille de trois octets et ne sont pas utilisés pour toutes les opérations. Le champ d'opération identifie les différentes lectures/écritures, le commencement de traitement pour un module, ainsi que la complétion d'une opération demandée précédemment. Les différentes opérations possibles sont définies au tableau 3.1.

Tableau 3.1 Définition des opérations possibles

Opération	Valeur	Description	Opérandes
READ_OP	0	Lecture d'une donnée d'un module	1 adresse mémoire; 1 dimension
WRITE_OP	1	Écriture d'une donnée vers un module	1 adresse mémoire; 1 dimension
START_OP	2	(L'arbitre) demande au module destination de commencer son traitement	1 adresse et 1 dimension : module d'entrée vidéo; 2 adresses et 1 dimension : ASIP et module de sortie vidéo
DONE_OP	3	Indique (à l'arbitre) que le module source a terminé son traitement pour l'opération précédemment demandée.	-

Dans le cas de l'opération « START_OP » pour le module d'entrée vidéo, l'adresse est le port pour l'image source. Pour le module de sortie vidéo, la première adresse est le port pour l'image source et la deuxième, pour l'image calculée. Dans le cas du processeur, la première adresse contient l'image précédente et la deuxième, l'image suivante. La dimension est toujours la taille des données traitées, lues ou écrites en octets.

3.1.1.3 Description modulaire

Cette section présente les particularités des modules qui forment le système réalisé. Il est à noter que les divers modules de conversion (*conv paq.*) des différents modules illustrés à la figure 3.1 ont été implémentés directement dans chacun des modules individuellement pour simplifier l'implémentation. Il s'agit en fait de la conversion des données en un paquet et vice-versa.

- *Mémoires*

Ce module contient un « sc_thread » qui exécute constamment une lecture au TI. Lorsqu'un paquet arrive, celui-ci est décodé pour identifier s'il s'agit d'une lecture ou d'une écriture. Un événement de lecture ou d'écriture est créé en fonction de cette identification afin d'aviser un autre « sc_thread ». S'il s'agit d'une lecture, le « sc_thread » de lecture décode l'adresse de lecture du paquet et la longueur du champ de données désirée, effectue la lecture d'un tableau C++, crée le paquet à envoyer avec toutes les données nécessaires, attend un cycle afin de simuler le cycle de lecture et, enfin, effectue l'écriture (bloquante) au TI. S'il s'agit d'une écriture, les données reçues sont simplement écrites dans le tableau C++ à l'adresse mémoire spécifiée. Ce module gère également les collisions de lecture/écriture avec des événements internes afin d'émuler le comportement réel d'une mémoire à un port.

- *Arbitre*

Ce module contient un « sc_thread » qui est en charge de créer et d'émettre les divers paquets de contrôle aux autres modules (avec l'opération « START_OP ») et d'attendre les confirmations d'opérations terminées (« DONE_OP »), provenant d'un module spécifique à un temps donné, afin d'arbitrer les échanges de données. Il suit un ordre d'exécution spécifique afin de réaliser l'opération de compensation de mouvement efficacement. Il détermine également quelles mémoires sont utilisées pour les différentes images successives.

- *Module d'entrée vidéo*

Ce module possède un « sc_thread » qui est continuellement en charge de vérifier si un paquet-requête arrive du TI. Dans un tel cas, un événement de lecture est appelé. Cet évènement active un deuxième « sc_thread » indépendant. Ce « sc_thread » ouvre un fichier contenant la prochaine image source d'une séquence à traiter et appelle une fonction en charge de créer des paquets de 256 octets contenant chacun une partie de cette image. L'adresse de destination (port) est incluse dans le paquet. Le paquet est ensuite envoyé à la mémoire par l'entremise du TI par cette fonction. Une fois que l'image est complètement transmise, une fonction en charge de transmettre un paquet de contrôle à l'arbitre est appelée. Le paquet indique que le module a terminé de transmettre l'image d'entrée. Le processus recommence lors de la réception d'un nouveau paquet-requête du TI.

- *Module de sortie vidéo*

Tout comme le module d'entrée vidéo, ce module possède un premier « sc_thread » qui est constamment en attente de réception d'un paquet. Le paquet reçu peut être une donnée lue (suite à une requête de ce même module de sortie vidéo) ou une requête visant à effectuer une transmission d'images à partir des mémoires vers la sortie.

S'il s'agit d'un paquet-requête qui demande la transmission des images, un événement de lecture est appelé. Cet évènement réveille un deuxième « sc_thread » indépendant. Ce deuxième « sc_thread » est en charge d'envoyer, une à la fois, les requêtes de lecture aux deux mémoires contenant les deux images concernées (une image source et une image calculée). Les adresses (ports) desquelles ont doit lire ces images sont contenues dans le paquet-requête. Les lectures se font par blocs de 256 octets et l'image source doit être lue en entier avant de débuter la lecture de l'image calculée. Il y a toujours une seule requête de lecture en attente à la fois. On attend de recevoir le paquet de données avant de générer la prochaine requête.

Par contre, si le premier thread reçoit des données à sortir, elles sont d'abord sauvegardées dans un tableau. Un événement est ensuite appelé afin d'activer le deuxième « sc_thread » discuté précédemment. Ceci est dans le but de générer le prochain paquet de requête des données à sortir. Lorsqu'une image est complètement lue, l'ensemble des données reçues pour cette image sont écrites dans un fichier. Lorsque les deux images sont complètement lues, un paquet de complétion est envoyé à l'arbitre pour l'aviser. Ceci est fait par le deuxième « sc_thread » qui appelle une fonction en charge de générer ce paquet.

- *Contrôleur d'ASIP et mémoire locale RAM (ASIP & RAM Ctrl.)*

Ce module décode les paquets reçus du TI, initie et effectue les transferts de données avec les mémoires locales, demande à l'ASIP de commencer son traitement et informe l'arbitre de la complétion de sa tâche. Il est le module qui gère toutes les opérations qui touchent directement à l'ASIP, à l'exception des calculs eux-mêmes qui sont dictés par le code qui s'exécute sur l'ASIP.

- *TI et module de collecte de données*

Ces modules sont les mêmes que ceux présentés au chapitre 2.

3.1.2 Collecte de données et calculs de statistiques

La collecte de données se fait avec les modules de collecte de données (« moniteur ») présentés au chapitre 2. L'analyse statistique est également faite avec les mêmes scripts Perl présentés au chapitre 2. Ceci montre l'harmonie entre les divers aspects de l'environnement créé.

3.1.3 Synthèse sur l'intégration d'une application concrète

Cette section a permis de montrer l'intégration d'une application réelle à l'environnement présenté au chapitre précédent. Un aspect complémentaire, abordé dans la prochaine section, est l'intégration d'un modèle de trafic synthétique existant.

3.2 Intégration d'un modèle de trafic synthétique à l'environnement

Bien que l'environnement à haut niveau avec trafic aléatoire contraint (ainsi que l'aisance d'y intégrer une application réelle matériel-logiciel) permette d'avoir une validation de performance efficace, un autre aspect important doit être considéré. Il s'agit de l'intégration d'un modèle de trafic synthétique réaliste à notre environnement SystemC. La présente section couvre cet aspect.

3.2.1 Concepts pour l'intégration d'un modèle de trafic synthétique

Les modèles de trafic synthétiques sont habituellement conçus à partir de statistiques tirées d'applications réelles; ils sont souvent déjà codés en C ou C++ et disponibles *a priori* ou via diverses sources comme l'Internet. Les modifications à appliquer à un modèle de trafic C++ pour le rendre compatible avec l'environnement SystemC sont étudiées dans cette section. Ceci permet de rendre disponible des modèles de trafic réalistes pour l'exploration architecturale et pour la vérification. Les modèles de TI conçus pour ce projet sont à nouveau réutilisés. La compagnie *Tundra Semiconductor*, qui a supporté ce projet, a fourni un tel modèle de trafic C++ pour ces travaux. Le travail décrit dans cette section a été réalisé de façon uniquement théorique. Les étapes réalisées sont l'analyse du code, l'élaboration de la stratégie pour intégrer le modèle de trafic dans un environnement SystemC et la conception d'un module de conversion d'interface non-bloquant. L'implémentation complète pourrait faire l'objet d'un projet ultérieur.

Pour commencer, le temps de cycle du modèle de trafic doit être synchronisé au temps du TI SystemC. Ceci peut se réaliser d'au moins deux façons. La première est d'ajouter un « sc_module » qui synchronise les événements du modèle de trafic et le temps de base du TI. La deuxième façon de procéder est de modifier le code du modèle de trafic pour l'encapsuler dans un « sc_module » et intégrer l'instruction « wait » de SystemC au modèle de trafic. La deuxième approche a été choisie parce qu'elle permet d'avoir plusieurs modules qui communiquent en parallèle de façon indépendante et utilise directement les fils d'exécution virtuels (« pseudo multi-threading ») de la norme SystemC IEEE 1666 [16].

Avec cette technique, le « wait » de SystemC doit être intégré au modèle pour synchroniser avec le « wait » du TI. Aussi, avec la deuxième méthode, le modèle de trafic fournit par *Tundra Semiconductor* a besoin d'être inclus dans un « sc_module » tel que présenté à la figure 3.2.

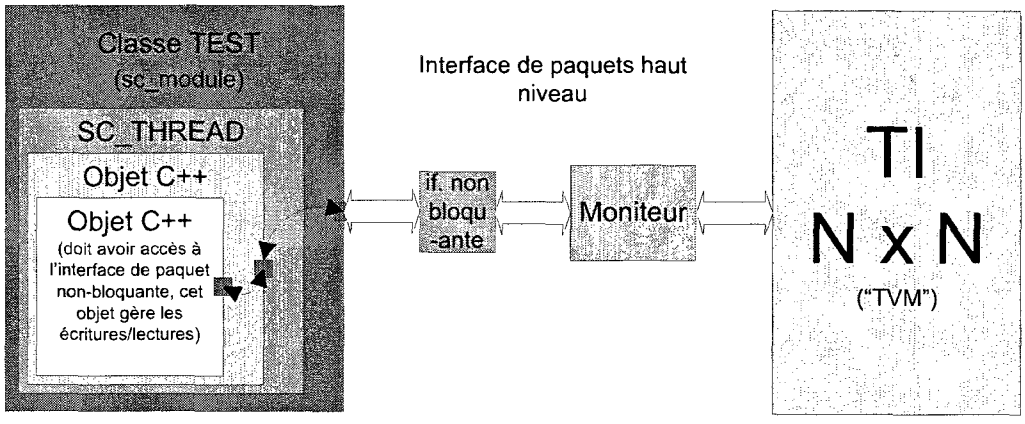

Figure 3.2 Environnement de vérification/validation avec modèle de trafic externe intégré

Cette figure représente une vue générale du travail à accomplir pour intégrer le modèle de trafic. Typiquement, un modèle externe C++ possède une hiérarchie de classes et doit accéder, à partir de ses sous-classes, au « sc_port » du module qui l'englobe pour permettre les communications avec un module externe. Tel qu'illustré par les flèches à l'intérieur de la classe « test » de la figure 3.2, ce « sc_port » peut être passé dans la hiérarchie de classe en utilisant des pointeurs. Ce modèle de trafic plus complexe englobe généralement toutes les lectures et écritures dans une même fonction/classe pour tous les ports d'entrée/sortie. Donc, les fonctions d'écriture/lecture du TI doivent redonner le contrôle lors de chaque appel par opposition aux tissus présentés au chapitre 2 qui ne redonnent le contrôle qu'une fois le paquet transmis. Cette conversion est faite dans un module *if. non-bloquante*. Il s'agit du module de conversion d'interface non-bloquant discuté brièvement au chapitre 2. Ce dernier a été réalisé et testé mais n'a pas été intégré au modèle de trafic synthétique. Le mécanisme décrit permet de synchroniser deux modules même si leur gestion du temps est *a priori* indépendante.

3.2.2 Module de conversion réalisé (if. non-bloquante)

Ce module n'étant qu'un intermédiaire, il possède et appelle les mêmes fonctions d'écriture, de lecture et d'attente que le module de test et le tissu d'interconnexion. À l'interne, ce module reçoit une requête de lecture/écriture, lance un « sc_thread » et redonne le contrôle au module de test. Ensuite, il y a un mécanisme qui indique si la requête a été acceptée ou non. Ceci est fait car le paquet prend plusieurs cycles à être transmis et bloque par conséquent l'accès d'un port pendant ces cycles. Pour une requête d'écriture avec succès, la réponse d'acceptation est aussitôt transmise et l'accès au port est refusé pour les prochaines requêtes d'écritures pendant la transmission du paquet accepté. Dans le cas d'une lecture avec succès, la lecture est enregistrée lors du premier appel et l'accès est tout de même refusé pour le présent et les prochains appels pendant le temps de transmission. Il est pris pour acquis que le module appelant interroge la lecture à chaque cycle jusqu'à l'acceptation de la requête. Le paquet est inclus avec la réponse d'acceptation lorsque le nombre de cycles écoulés depuis la requête correspond au temps de transmission du paquet. Il est aussi possible qu'aucun paquet ne soit prêt pour une lecture. Dans ce cas, le contrôle est redonné à la fonction appelante avec une réponse indiquant que la lecture a échouée.

3.2.3 Synthèse sur l'intégration d'un modèle de trafic synthétique

Même si cette section contient en grande partie des aspects théoriques, les travaux ont tout de même permis de montrer l'aisance à intégrer des modèles de trafic existants à l'environnement créé. Ceci peut devenir un aspect important dans une industrie pour laquelle recommencer du début la création de modèles de trafic, en même temps que d'envisager une nouvelle approche de validation et de vérification, pourrait devenir un obstacle. L'intégration de modèles de trafic existants est un élément complémentaire qui permet de rendre l'environnement plus complet.

3.3 Conclusion

Ce chapitre a permis de montrer l'intégration d'une application réelle à l'environnement présenté au chapitre précédent. Il a également permis de montrer l'intégration d'un modèle de trafic synthétique. Le prochain chapitre montre quelques résultats obtenus par simulation avec l'environnement discuté au chapitre 2.

CHAPITRE 4. RÉSULTATS

Le but du projet est de créer une plateforme de vérification et validation de performance complète et aisément configurable. Le but n'est donc pas de prouver des propriétés spécifiques d'un TI. Cependant, afin de montrer l'utilisation de l'environnement, le présent chapitre montre les résultats statistiques obtenus pour quelques scénarios spécifiques. Ce chapitre inclut d'abord une description sommaire des étapes de la méthode de conception que l'on considère comme un résultat des travaux.

4.1 Méthode de conception qui englobe la validation

Un des résultats de cette recherche est une méthode de conception enrichie qui englobe des éléments de validation des performances. Cette méthode a émergé suite aux diverses lectures et à l'expérience acquise lors de la réalisation de l'environnement présenté aux chapitres précédents. Les principales étapes de la méthode de développement proposée et utilisée s'énoncent comme suit :

1- Étudier le système à réaliser et élaborer une architecture potentielle (avec ses interrogations);
2- Réaliser une implémentation de l'architecture à haut niveau d'abstraction;
3- Élaborer un modèle de trafic, de création de base de données et d'analyse statistique approprié;
4- Réaliser des tests et effectuer des simulations simples afin de prouver que l'implémentation est adéquate et que les résultats sont plausibles (avec des corrections de fonctionnement au besoin);
5- Réaliser des tests et effectuer des simulations plus complexes (avec un modèle de haut niveau, une application complète matérielle-logicielle et un modèle de trafic synthétique) pour valider que l'architecture obtient

bien la performance voulue (avec des corrections architecturales, au besoin);

6- Créer les modules de conversions nécessaires, les intégrer à l'environnement de validation (pour former le banc d'essai) et implémenter, en langage matériel, l'architecture développée;
7- Effectuer une vérification matérielle simple avec le TI matériel et le banc d'essai;
8- Valider que l'implémentation matérielle obtient bien les performances prévues avec les modèles de trafic disponibles (itérations possibles, mais peu probables avec une implémentation adéquate du modèle de haut niveau);
9- Effectuer une vérification/validation avancée de l'implémentation matérielle avec de nouveaux tests pour augmenter la couverture de la vérification/validation.

Cette méthode est une des avenues possibles qui permet d'optimiser les diverses étapes à la conception d'un TI ou d'un circuit intégré divers. Lors d'un projet, la productivité est ainsi normalement améliorée (des cas d'exception peuvent se présenter). Les deux principales sources de ce gain de productivité, pour l'environnement présenté, sont l'intégration de la simulation (pour valider les performances) et la méthode unifiée proposée. Cette dernière utilise un langage de vérification avancé dans le but d'intégrer la validation de performance et la vérification dans un même environnement.

Cette section a montré les diverses étapes à suivre pour le développement d'un CI divers tout en appliquant la méthode intégrant la validation de performance proposée par le présent projet. La prochaine section montre des exemples de résultats obtenus lors de la simulation avec l'environnement présenté au chapitre 2.

4.2 Résultats obtenus avec l'environnement à haut niveau

Les résultats de divers essais réalisés avec l'environnement de validation à haut niveau présenté au chapitre 2 sont présentés dans cette section. Cela inclut les résultats des analyses pour un trafic symétrique et asymétrique ainsi que des simulations pour caractériser le comportement d'un TI.

Les résultats présentés sont donc tous obtenus en utilisant uniquement des trafics pseudo-aléatoires contraints tels que ceux décrits aux chapitres 1 et 2. Aucun résultat n'est donc présenté concernant l'application réelle s'exécutant sur un ASIP qui permet un trafic réel. L'analyse de la simulation de ce dernier environnement a permis de prouver le but premier, c'est-à-dire, le bon fonctionnement suite à l'intégration de notre environnement avec TI à un ASIP en utilisant XTSC.

4.2.1 Aspect commun aux simulations : modèle de paquet

Un aspect commun aux simulations de cette section est présenté ici. Il s'agit du modèle de paquet. Les valeurs des différents paramètres génériques qui décrivent les dimensions des paramètres des paquets sont définies au tableau 4.1. La description détaillée de chacun de ces paramètres est présentée au tableau 2.1.

Tableau 4.1 Définition des dimensions du paquet

Générique	Valeur (bits)
ADDR_W	8
PACK_ID_W	14
PAYL_W	8
DATA_WD_W	Variable

4.2.2 Détermination du facteur d'accélération et analyse pour un trafic symétrique

Avec l'architecture de TI présentée au chapitre 2, pour des trafics approchant la limite des ports d'entrée, il est improbable de réussir à obtenir un trafic approchant la pleine capacité des liens de sortie sans avoir un facteur d'accélération qui est défini comme le rapport entre la bande passante disponible à l'intérieur d'un TI sur le débit de pointe de ses ports. En effet, les chemins de données formant un commutateur crossbar, dès qu'un port d'entrée envoie vers une sortie, le commutateur crossbar a pour effet de bloquer d'autres transmissions potentielles. De plus, les VOQ n'ont pas toujours au moins un paquet en banque qui utilise les chemins de données libres dans le commutateur crossbar. Donc, un facteur d'accélération permet d'accélérer les transmissions à l'interne, ce qui a pour effet de libérer des connexions plus rapidement et qui permet potentiellement d'améliorer l'utilisation de la bande passante en sortie. Ultimement, cela permet d'éviter la perte de paquets en cas de pointe de trafic pour une période significative. Par contre, un facteur d'accélération des chemins de données augmente significativement la taille de silicium vu le grand nombre de fils impliqués. Il est donc important de bien choisir le facteur d'accélération. L'utilisation d'un modèle comme celui proposé permet une telle validation du facteur d'accélération. Pour ce faire, cette section propose d'analyser des simulations avec différents facteurs d'accélération pour des TI de 4, 16 et 32 ports pour un trafic élevé mais symétrique en entrée. Il est évidemment possible qu'un facteur d'accélération plus élevé soit nécessaire pour des trafics asymétriques élevés; ceci nécessiterait une autre étude avec un trafic en conséquence. L'analyse est faite indépendamment avec les deux algorithmes d'ordonnanceurs présentés, soit le WRR et le WWFA.

L'analyse se fait avec des paquets de tailles variables pour simuler un cas réel. La taille utilisée doit normalement être choisie en fonction de la connaissance qu'on a des tailles des paquets qui sont habituellement transmis dans le TI. Ceci dépend, entre autres, du protocole de communication utilisé et des données transmises. Pour les simulations de cette section, on utilise des tailles des champs de données variant de 50 à 90 octets de

façon aléatoire. Pour un grand nombre de cycles de simulation, tous les ports transmettent des paquets de tailles moyennes statistiquement égales. Le trafic aux entrées est poussé à la limite de cent pour cent sur toutes les entrées. La destination est toujours aléatoire. La taille des FIFO VOQ internes est fixée à 10 paquets. Lors de l'initialisation, on laisse les VOQ se remplir jusqu'à une taille moyenne de 5 paquets avant d'activer les sorties (afin d'atteindre le régime permanent d'un grand trafic plus rapidement). L'analyse statistique commence après *150* nombre de ports du TI* cycles de simulation (total), ce qui permet de s'assurer d'être en régime permanent. Cette valeur dépend du nombre de ports puisque, dans le cas d'une architecture avec VOQ, le temps nécessaire au remplissage des files est proportionnel au nombre de files. L'analyse se fait pour, en moyenne, 10 000 cycles par port. On s'assure évidemment que la simulation est toujours en régime permanent à ce moment. Les tableaux 4.2, 4.3 et 4.4 détaillent les paramètres de configuration pour le TI, la génération du trafic aléatoire et l'analyse statistique.

Tableau 4.2 Configuration du TI utilisé pour déterminer le facteur d'accélération

Champ	Valeur
Type d'ordonnanceur	WRR vs WWFA
Nombre de ports du TI	4, 16 et 32
Facteur d'accélération du commutateur crossbar	1.0 à 1.5 (par incréments de 0.05 ou 0.025)
TI SystemC vs RTL	TI SystemC
Poids par port pour l'ordonnanceur WRR	1 pour tous les ports
Profondeur des files	10 paquets
Profondeur des FIFO de sortie	2 paquets
Largeur des ports	56 bits
Période d'horloge	4 ns
Nombre de cycles d'initialisation	0 cycle
Nombre de paquet moyen par file VOQ (initialisation)	5 paquets

Tableau 4.3 Paramètres de la génération du trafic pour déterminer le facteur d'accélération

Nom du paramètre	valeur/ unité	Valeur
NB_PACKETS	Nb. paquets	10 000
RANDOM_DEST	on/off	On
BW_MIN_DL	Nb. octets	50
BW_MAX_DL	Nb. octets	90
BW_PERC_BW	0-100 %	100%
MUL_FACTOR_BW_PERC_BW [NB_PORT]	Ratio	1
MUL_FACTOR_BW_DL [NB_PORT]	Ratio	1
N_CONSEC	Entier	2
SCV	Interrupteur	On
NB_PACKET_SAME_DEST	Entier	1

Tableau 4.4 Paramètres de l'analyse statistique pour déterminer le facteur d'accélération

Nom du paramètre	Valeur
$NB_INIT_CYCLES	*150*nombre de ports*
$NB_TOT_DATACYCL_ANAL	10 000 par port
$NB_CYCLE_MAX_IDLE	200

Pour cette analyse, on veut amener à 1.0 le taux d'utilisation normalisé des liens aux sorties (avec un facteur d'accélération réaliste). Les tableaux 4.5, 4.6 et 4.7 résument les taux d'utilisation totaux des ports de sortie obtenus après analyse avec un script Perl des différents résultats de simulation. La figure 4.1 montre l'évolution du taux d'utilisation total des ports de sortie en fonction du facteur d'accélération pour les deux algorithmes d'ordonnanceur et les trois tailles de TI.

Tableau 4.5 Taux d'utilisation des ports de sortie pour un TI à 4 ports

Algorithme d'ordonnancement	Facteur d'accélération	Taux d'utilisation des ports de sortie
WRR	1.00	0.842
	1.05	0.887
	1.10	0.947
	1.15	0.973
	1.20	0.993
	1.25	0.996
WWFA	1.00	0.845
	1.05	0.889
	1.10	0.938
	1.15	0.987
	1.20	0.992
	1.30	0.995

Tableau 4.6 Taux d'utilisation des ports de sortie pour un TI à 16 ports

Algorithme d'ordonnancement	Facteur d'accélération	Taux d'utilisation des ports de sortie
WRR	1.000	0.979
	1.025	0.994
	1.050	1.000
WWFA	1.000	0.981
	1.025	0.997
	1.050	1.000

Tableau 4.7 Taux d'utilisation des ports de sortie pour un TI à 32 ports

Algorithme d'ordonnancement	Facteur d'accélération	Taux d'utilisation des ports de sortie
WRR	1.000	0.996
	1.025	1.000
WWFA	1.000	0.995
	1.025	1.000

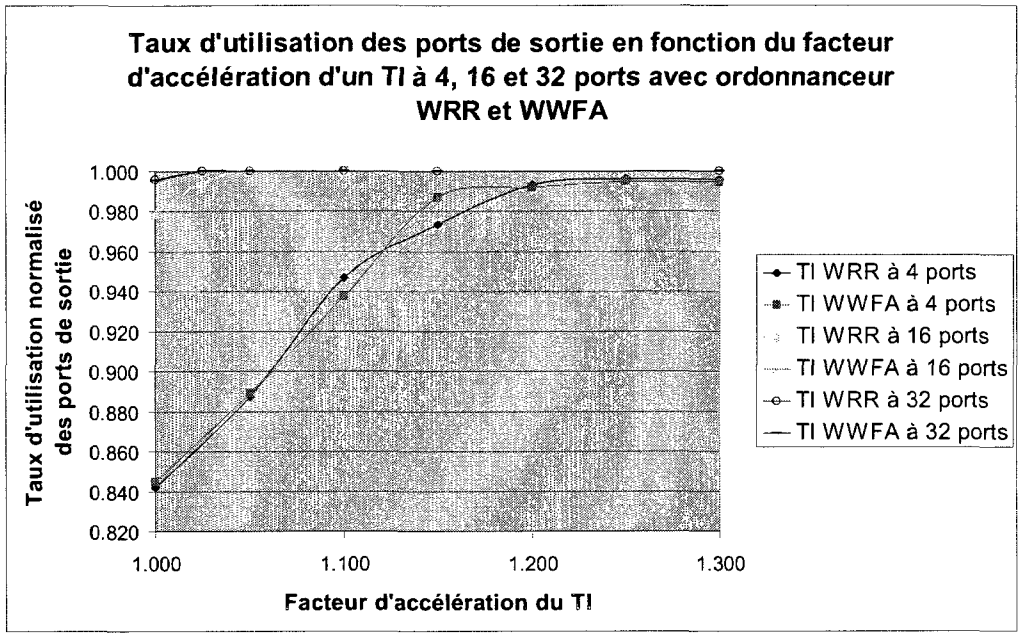

Figure 4.1 Taux d'utilisation des ports de sortie en fonction du facteur d'accélération

On constate que, pour les 2 ordonnanceurs, un facteur d'accélération de 1.2 semble un bon compromis pour un TI de 4 ports. Il est possible d'augmenter légèrement les performances avec un facteur d'accélération de 1.25, mais le gain est faible compte tenu de la surface supplémentaire de silicium utilisée pour ce dernier facteur d'accélération. Pour le scénario testé, il n'y a aucun gain à avoir un facteur d'accélération supérieur à 1.25. Les 2 algorithmes d'ordonnancement se comportent en général de façon similaire pour un même facteur d'accélération. Cependant, pour un facteur d'accélération de 1.15, l'algorithme WWFA semble plus performant que le WRR. Cela peut devenir un facteur à considérer lorsque vient le temps de choisir l'algorithme d'un ordonnanceur dépendamment du facteur d'accélération choisi.

Pour un TI à 16 ports, on constate qu'un facteur d'accélération bien moins élevé est nécessaire pour atteindre une même performance. En effet, un facteur d'accélération

de 1.05 permet une performance idéale avec un taux d'utilisation des sorties de 1.0 pour le scénario de simulation du présent exercice. On observe également que le WWFA est légèrement plus performant que le WRR pour un facteur d'accélération inférieur à 1.05. Pour le scénario testé, il n'y a aucun gain à avoir un facteur d'accélération supérieur à 1.05.

Avec un TI à 32 ports, on constate que même un facteur d'accélération de 1.0 donne de très bonnes performances. On a besoin d'un facteur d'accélération de seulement 1.025 pour obtenir une utilisation maximale des ports de sortie. Pour le scénario testé, il n'y a aucun gain à avoir un facteur d'accélération supérieur à 1.025. Les deux ordonnanceurs performent de façon quasiment identique de sorte qu'il est difficile de distinguer les deux courbes de la figure 4.1.

On note que, de façon générale, plus la taille d'un TI est élevée, plus le facteur d'accélération nécessaire en cas de trafic symétrique est faible. Il y a là un point intéressant pour les TI de prochaines générations qui s'avèrent avoir de plus en plus de ports. Ceci est pertinent puisque la collaboration avec le partenaire industriel a semblé montrer qu'il est, pour celui-ci, acquis qu'un facteur d'accélération est requis. Or, selon nos résultats, le facteur d'accélération n'offre pas de grands bénéfices pour des TI avec un grand nombre de ports.

Par contre, les taux d'utilisation des liens de sortie de la figure 4.1 sont ceux obtenus alors que les files VOQ sont profondes de 10 paquets. Il serait intéressant de pousser l'analyse plus loin en y intégrant le facteur d'accélération nécessaire en fonction de la taille de ces files.

Dans ce qui suit, on présente les résultats les plus pertinents comme référence d'une analyse de trafic symétrique pour les facteurs d'accélération qui semblent les meilleurs compromis pour chaque taille de TI. Ces résultats sont particulièrement

intéressants afin de comparer aux résultats de l'analyse d'équité avec un trafic asymétrique de la section suivante, et ce, pour les 2 ordonnanceurs réalisés. Cela permet d'observer la différence de comportement d'un TI lorsqu'il est soumis à un trafic symétrique ou asymétrique. Les tableaux 4.8 à 4.13 montrent les principaux résultats obtenus suite à une analyse, avec un script Perl, des diverses simulations réalisées. Il est à noter que, tel que montré au chapitre 1, un paramètre de Jain se rapprochant de 1.0 est considéré équitable pour la statistique analysée. Également, deux paramètres de Jain sont calculés. Le premier est celui qui identifie l'équité entre tous les ports. Le deuxième représente l'équité du dernier port (avec de plus gros paquets pour l'analyse asymétrique de la section 4.2.3) versus la moyenne des valeurs obtenues pour les autres ports (avec une taille de paquet uniforme). Ce dernier permet de faire ressortir davantage ce qui nous intéresse pour l'analyse asymétrique de la section 4.2.3, c'est-à-dire, le débalancement entre le dernier et les autres ports. Ceci est surtout vrai lorsqu'il y a un grand nombre de ports car le débalancement pour le dernier port est alors moins visible avec le premier paramètre de Jain.

Tableau 4.8 Statistiques de simulation d'un trafic symétrique sur un TI 4x4 avec ordonnanceur WRR et un facteur d'accélération de 1.2

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus	Équité de connexion	Occupation des files (VOQ)	Latence moyenne	$99^{ième}$ percentile latence	Taux d'utilisation normalisé du lien
Type de ports	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	874	9898	24	83.0	32.1	154	463	0.988
Port # 1	891	10087	8	84.5	32.5	161	471	0.990
Port # 2	880	9977	19	83.7	29.1	142	387	0.999
Port # 3	889	10045	13	84.2	31.2	153	426	0.995
Total	3534	10067	64	-	26.2	153	446	0.993
Jain 1 (tous)	-	-	0.8752	1.0000	0.9982	0.9980	0.9943	-
Jain 2 (dernier)	-	-	1.0000	1.0000	1.0000	1.0000	0.9997	-

Tableau 4.9 Statistiques de simulation d'un trafic symétrique sur un TI 4x4 avec ordonnanceur WWFA et un facteur d'accélération de 1.2

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus	Équité de connexion	Occupation des files (VOQ)	Latence moyenne	$99^{ième}$ percentile latence	Taux d'utilisation normalisé du lien
Type de ports	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	879	9938	20	83.2	32.4	156	466	0.984
Port # 1	900	10192	3	85.3	26.2	133	485	0.987
Port # 2	861	9741	40	81.6	36.0	169	383	0.997
Port # 3	898	10136	4	84.8	25.4	129	364	0.999
Total	3538	10083	67	-	24.5	146	442	0.992
Jain 1 (tous)	-	-	0.5542	0.9997	0.9790	0.9877	0.9852	-
Jain 2 (dernier)	-	-	1.0000	0.9999	0.9885	0.9930	0.9901	-

Tableau 4.10 Statistiques de simulation d'un trafic symétrique sur un TI 16x16 avec ordonnanceur WRR et un facteur d'accélération de 1.05

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus	Équité de connexion	Occupation des files (VOQ)	Latence moyenne	99ième percentile latence	Taux d'utilisation normalisé du lien
Type de ports	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	886	9996	25	95.7	36.8	675	2290	1.000
Port # 1	891	10142	23	96.8	39.2	719	1746	1.000
Port # 2	889	10022	28	95.7	35.1	630	1939	1.000
Port # 3	893	10083	21	96.2	37.3	671	1940	1.000
Port # 4	878	9989	30	95.5	38.8	709	1693	1.000
Port # 5	877	9912	28	94.7	36.8	665	1680	1.000
Port # 6	880	9974	21	95.3	34.7	635	1602	1.000
Port # 7	878	9928	25	94.6	36.5	661	1630	1.000
Port # 8	881	9914	31	94.4	39.7	710	1699	1.000
Port # 9	870	9916	24	94.7	39.3	730	1605	1.000
Port # 10	901	10097	32	96.2	36.9	655	1848	1.000
Port # 11	880	9911	33	94.7	33.8	602	1602	1.000
Port # 12	890	10028	17	95.8	38.3	696	1702	1.000
Port # 13	895	10087	24	95.8	38.7	705	1762	1.000
Port # 14	884	10002	27	95.7	36.8	667	1502	1.000
Port # 15	890	10002	28	95.4	37.6	662	1830	1.000
Total	14163	10001	417	-	32.8	675	1835	1.000
Jain 1 (tous)	-	-	*0.9741*	*1.0000*	*0.9980*	*0.9975*	*0.9893*	-
Jain 2 (dernier)	-	-	1.0000	1.0000	1.0000	0.9999	0.9995	-

Tableau 4.11 Statistiques de simulation d'un trafic symétrique sur un TI 16x16 avec ordonnanceur WWFA et un facteur d'accélération de 1.05

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus	Équité de connexion	Occupation des files (VOQ)	Latence moyenne	99ième percentile latence	Taux d'utilisation normalisé du lien
Type de ports	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	882	9961	23	95.3	38.8	709	2036	1.000
Port # 1	879	9996	24	95.4	38.9	696	1722	1.000
Port # 2	886	9990	25	95.4	33.3	597	1889	1.000
Port # 3	875	9894	36	94.5	34.9	611	1614	1.000
Port # 4	871	9913	28	94.8	39.6	717	1703	1.000
Port # 5	892	10054	25	96.1	36.4	664	1785	1.000
Port # 6	899	10156	16	97.0	39.7	729	2244	1.000
Port # 7	891	10087	26	96.2	37.8	692	1619	1.000
Port # 8	887	9980	32	95.0	37.2	663	1699	1.000
Port # 9	861	9826	35	93.8	37.3	670	1638	1.000
Port # 10	891	9981	31	95.2	35.0	626	1726	1.000
Port # 11	886	9942	32	95.0	34.1	607	1608	1.000
Port # 12	886	9978	25	95.3	37.6	685	1585	1.000
Port # 13	895	10081	21	95.8	41.2	751	1872	1.000
Port # 14	893	10109	23	96.7	36.9	666	1815	1.000
Port # 15	894	10070	16	96.1	40.1	728	1737	1.000
Total	14168	10002	418	-	33.1	676	1875	1.000
Jain 1 (tous)	-	-	0.9536	0.9999	0.9966	0.9955	0.9909	-
Jain 2 (dernier)	-	-	1.0000	1.0000	0.9986	0.9984	0.9999	-

Tableau 4.12 Statistiques de simulation d'un trafic symétrique sur un TI 32x32 avec ordonnanceur WRR et un facteur d'accélération de 1.2

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus	Équité de connexion	Occupation des files (VOQ)	Latence moyenne	99ième percentile latence	Taux d'utilisation du lien
Type	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	883	9951	47	97.9	41.4	1463	4062	1.000
Port # 1	882	10053	30	98.6	42.8	1554	3760	1.000
Port # 2	892	10062	27	98.8	36.6	1333	3118	1.000
Port # 3	886	10023	47	98.4	41.1	1456	4040	1.000
Port # 4	888	10082	42	98.9	42.6	1513	3704	1.000
Port # 5	886	10040	31	98.6	46.3	1641	3711	1.000
Port # 6	880	9982	34	97.9	41.5	1488	3761	1.000
Port # 7	894	10145	38	99.7	39.5	1403	3680	1.000
Port # 8	899	10073	36	98.9	41.0	1444	4182	1.000
Port # 9	876	10013	35	98.4	37.1	1331	3083	1.000
Port # 10	894	10049	29	98.6	40.6	1430	3762	1.000
Port # 11	892	10045	39	98.6	38.8	1379	3327	1.000
Port # 12	886	10000	29	98.2	40.7	1440	3549	1.000
Port # 13	888	10040	33	98.7	40.3	1462	3683	1.000
Port # 14	887	10056	33	98.8	38.8	1415	3520	1.000
Port # 15	886	10011	45	98.3	39.0	1371	3468	1.000
Port # 16	892	10134	37	99.4	38.3	1379	3423	1.000
Port # 17	894	10119	32	99.3	37.9	1346	3730	1.000
Port # 18	888	10046	33	98.8	41.6	1481	3683	1.000
Port # 19	884	10003	39	98.3	39.6	1406	3421	1.000
Port # 20	883	9943	35	97.5	40.0	1385	3314	1.000
Port # 21	886	9976	27	98.0	41.2	1458	3349	1.000
Port # 22	886	10002	24	98.2	41.1	1455	3515	1.000
Port # 23	872	9866	30	96.9	42.2	1495	3087	1.000
Port # 24	869	9804	39	96.2	40.7	1428	3112	1.000
Port # 25	862	9798	42	96.0	40.4	1411	3026	1.000
Port # 26	877	9932	34	97.5	36.3	1298	3161	1.000
Port # 27	867	9811	40	96.4	39.4	1410	3249	1.000
Port # 28	879	9954	28	97.9	41.8	1509	3687	1.000
Port # 29	881	9935	33	97.6	39.1	1387	3484	1.000
Port # 30	891	10038	38	98.6	39.5	1368	3605	1.000
Port # 31	886	10018	51	98.5	41.4	1405	3366	1.000
Total	28296	10002	1137	-	35.4	1430	3786	1.000
Jain 1	-	-	0.9690	0.9999	0.9977	0.9977	0.9932	-
Jain 2	-	-	1.0000	1.0000	0.9998	0.9999	0.9995	-

Tableau 4.13 Statistiques de simulation d'un trafic symétrique sur un TI 32x32 avec ordonnanceur WWFA et un facteur d'accélération de 1.2

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus	Équité de connexion	Occupation des files (VOQ)	Latence moyenne	$99^{ième}$ percentile latence	Taux d'utilisation du lien
Type	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	890	10080	30	99.2	39.3	1388	3804	1.000
Port # 1	871	9962	52	97.6	38.4	1382	3416	1.000
Port # 2	888	10009	38	98.3	41.8	1484	3798	1.000
Port # 3	887	10056	35	98.8	41.2	1467	3701	1.000
Port # 4	874	9932	35	97.5	40.0	1412	3366	1.000
Port # 5	888	10053	23	98.7	40.3	1450	3528	1.000
Port # 6	891	10104	24	99.1	40.8	1477	4133	1.000
Port # 7	875	9924	41	97.5	40.8	1445	3221	1.000
Port # 8	891	10008	28	98.2	41.4	1486	3502	1.000
Port # 9	876	9997	35	98.3	37.0	1292	3259	1.000
Port # 10	892	10044	35	98.6	40.5	1441	4085	1.000
Port # 11	893	10048	28	98.6	41.0	1466	4072	1.000
Port # 12	875	9893	33	97.3	40.6	1421	3206	1.000
Port # 13	895	10100	28	99.2	38.3	1358	3579	1.000
Port # 14	882	9992	53	98.2	36.8	1274	3440	1.000
Port # 15	888	10033	40	98.5	41.2	1444	3578	1.000
Port # 16	884	10066	32	98.7	39.5	1396	3471	1.000
Port # 17	874	9884	43	97.0	36.4	1274	3056	1.000
Port # 18	875	9865	51	97.1	38.9	1371	3233	1.000
Port # 19	877	9916	55	97.5	40.0	1368	3392	1.000
Port # 20	890	10022	22	98.4	41.3	1468	3548	1.000
Port # 21	890	10054	19	98.7	39.0	1406	3299	1.000
Port # 22	884	10016	37	98.4	39.3	1387	3728	1.000
Port # 23	879	9948	45	97.7	41.0	1455	3438	1.000
Port # 24	874	9874	50	97.0	40.4	1368	3320	1.000
Port # 25	875	9933	38	97.4	41.0	1441	3438	1.000
Port # 26	889	10033	33	98.5	40.8	1441	3882	1.000
Port # 27	891	10060	16	99.0	41.6	1508	4051	1.000
Port # 28	881	9924	31	97.5	41.5	1477	3308	1.000
Port # 29	883	9911	40	97.3	37.7	1321	3312	1.000
Port # 30	894	10083	31	99.1	41.2	1473	3779	1.000
Port # 31	904	10181	22	100.0	40.7	1456	3612	1.000
Total	28300	10001	1123	-	34.8	1416	3741	1.000
Jain 1	-	-	0.9248	0.9999	0.9987	0.9981	0.9938	-
Jain 2	-	-	1.0000	0.9999	0.9999	0.9998	0.9999	-

On observe, en général, que les diverses statistiques sont relativement uniformes entre les divers ports d'entrée/sortie. Les paramètres de Jain montrent également cette uniformité. En se basant sur les facteurs de Jain, il ne semble pas y avoir de différence d'équité entre le WRR et le WWFA pour des TI de 16 et 32 ports lorsque le trafic est symétrique. Pour des TI de 4 ports, le WRR semble légèrement plus équitable. On peut conclure ceci non seulement à cause des paramètres de Jain qui ont des valeurs plus faibles pour le WWFA, mais aussi parce qu'on observe un débalancement des paramètres pour les différents ports. De façon générale, les TI avec ordonnanceur WWFA ont une latence et un $99^{ième}$ percentile de la latence légèrement plus faible que les TI avec un ordonnanceur WRR. Cela peut mener à la conclusion inverse que le WWFA est plus équitable. Par contre, la différence est négligeable. Donc, on ne peut conclure qu'un algorithme est plus équitable que l'autre pour un trafic symétrique.

Il a été montré, dans cette section, que l'environnement de validation de performance développé permet de faire une analyse sur un trafic symétrique et d'en conclure les facteurs d'accélération nécessaires. La section suivante effectue une analyse d'équité pour un trafic asymétrique.

4.2.3 Analyse de l'équité pour un trafic asymétrique

En plus du facteur d'accélération, il existe d'autres paramètres d'architecture qui nécessitent des simulations afin de faire de bons choix. L'équité d'un algorithme est l'une des caractéristiques importantes d'un ordonnanceur. Il est donc pertinent de valider l'équité avec diverses simulations avant l'implémentation matérielle. La section présente montre un exemple d'analyse d'équité fait avec l'environnement créé pour un trafic asymétrique. L'analyse est faite dans le but de comparer l'équité des deux ordonnanceurs réalisés. Les divers paramètres d'équité discutés au chapitre 1 sont montrés et analysés pour les simulations de cette section.

Un trafic très asymétrique avec des paquets en moyenne 5 fois plus gros pour un des ports d'entrée est utilisé. Le dernier port (port 3 pour un TI 4x4; port 15 pour un TI 16x16; port 31 pour un TI 32x32) reçoit des paquets de tailles plus importantes. Ceci a comme but de créer un scénario de trafic asymétrique et d'observer l'équité des décisions de l'ordonnanceur malgré ce débalancement. Il s'agit de pousser l'algorithme dans des conditions limites. Une taille de paquet de base de 50 à 90 octets est conservée pour les autres ports, ce qui implique une taille de 250 à 450 octets pour le port qui reçoit des paquets d'une taille plus élevée. L'asymétrie est grande dans le but d'obtenir des résultats relativement exagérés, mais facile d'interprétation. L'intensité du trafic en entrée demeure à 100 % afin d'obtenir une situation de trafic élevé. Le facteur d'accélération utilisé est fixé à celui prouvé comme satisfaisant pour un trafic symétrique à la section précédente.

La destination est, encore une fois, aléatoire pour toutes les entrées. La taille des VOQ internes est fixée à l'équivalent de 100 paquets de base (c'est-à-dire de taille moyenne entre 50 et 90 octets) dans le but de simuler des files infinies. Lors de l'initialisation, on laisse les VOQ se remplir jusqu'à une taille moyenne de 20 paquets avant d'activer les sorties (afin d'atteindre le régime permanent d'un grand trafic plus rapidement). L'analyse statistique commence après *(600* nombre de ports du TI)* cycles de simulation (total), permettant de s'assurer d'être en régime permanent. L'analyse se fait pour, en moyenne, 10 000 cycles par port. On s'assure évidemment que la simulation est toujours en régime permanent au moment de terminer l'analyse. Les tableaux 4.14 à 4.16 détaillent les paramètres de configuration pour l'analyse statistique, le TI et la génération du trafic aléatoire.

Tableau 4.14 Paramètres de l'analyse statistique pour l'évaluation d'équité

Nom du paramètre	Valeur
$NB_INIT_CYCLES	*600* nombre de ports du TI*
$NB_TOT_DATACYCL_ANAL	10 000 par port
$NB_CYCLE_MAX_IDLE	200

Tableau 4.15 Configuration du TI utilisé pour l'évaluation d'équité

Champ	Valeur
Type d'ordonnanceur	WRR vs WWFA
Nombre de ports du TI	4, 16 et 32
Facteur d'accélération du commutateur crossbar	1.2 pour 4 ports; 1.05 pour 16 ports; 1.025 pour 32 ports
TI SystemC vs RTL	TI SystemC
Poids par port pour l'ordonnanceur WRR	5 pour tous les ports sauf le dernier port qui est à 1
Profondeur des files	100 paquets de base (i.e. des paquets de taille moyenne entre 50 et 90 octets, ce qui équivaut à une profondeur de 20 paquets pour les ports avec paquets de longueur 5x)
Profondeur des FIFO de sortie	2 paquets
Largeur des ports	56 bits
Période d'horloge	4 ns
Nombre de cycles d'initialisation	0 cycle
Nombre de paquet moyen par file VOQ (initialisation)	20 paquets

Tableau 4.16 Paramètres de la génération du trafic pour l'évaluation d'équité

Nom du paramètre	valeur/ unité	Valeur
NB_PACKETS	Nb. paquets	10 000
RANDOM_DEST	on/off	On
BW_MIN_DL	Nb. octets	50 ou 250 (dernier port)
BW_MAX_DL	Nb. octets	90 ou 450 (dernier port)
BW_PERC_BW	0-100 %	100%
MUL_FACTOR_BW_PERC_BW [NB_PORT]	Ratio	1
MUL_FACTOR_BW_DL [NB_PORT]	Ratio	5 pour le dernier port, 1 pour les autres
N_CONSEC	Entier	2
SCV	Interrupteur	On
NB_PACKET_SAME_DEST	Entier	1

Les tableaux 4.17 à 4.22 montrent les statistiques obtenues suite aux simulations. L'ensemble des résultats est présenté pour les deux ordonnanceurs et pour un TI de 4, 16 et 32 ports. Il est à noter que, tel que discuté à la section précédente, deux paramètres de Jain sont calculés. Le premier est celui qui identifie l'équité entre tous les ports. Le deuxième représente l'équité du dernier port (avec paquets plus gros) versus la moyenne des valeurs obtenues pour les autres ports (avec une taille de paquet uniforme). Ce dernier permet de faire ressortir davantage ce qui nous intéresse, c'est-à-dire, le débalancement entre le dernier et les autres ports. Ceci est surtout vrai lorsqu'il y a un grand nombre de ports, car le débalancement pour le dernier port est alors moins visible avec le premier paramètre de Jain.

Tableau 4.17 Statistiques de l'analyse à trafic asymétrique d'un TI 4x4 avec ordonnanceur WRR et un facteur d'accélération de 1.2

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus (1)	Équité de connexion (2)	Occupation des files (VOQ) (3)	Latence moyenne (4)	$gg^{ième}$ percentile latence (5)	Taux d'utilisation normalisé du lien
Type de ports	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	865	9778	0	75.7	27.8	1249	3393	0.925
Port # 1	892	10158	0	78.6	22.9	1041	2965	0.911
Port # 2	896	10122	0	78.4	19.2	853	1451	0.906
Port # 3	192	9946	0	76.3	30.1	1240	514	0.927
Total	2845	10902	0	-	24.6	1059	3188	0.917
Paramètre de Jain (tous)	-	-	1	0.9997	0.972	0.9784	0.7639	-
Paramètre de Jain (dernier)	-	-	1.0000	0.9999	0.9840	0.9930	0.6901	-

Tableau 4.18 Statistiques de l'analyse à trafic asymétrique d'un TI 4x4 avec ordonnanceur WWFA et un facteur d'accélération de 1.2

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus (1)	Équité de connexion (2)	Occupation des files (VOQ) (3)	Latence moyenne (4)	$99^{ième}$ percentile latence (5)	Taux d'utilisation normalisé du lien
Type de ports	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	846	9535	0	80.3	25.1	1146	2288	0.999
Port # 1	876	9964	0	84.0	14.5	672	1529	1.000
Port # 2	915	10329	0	87.2	7.7	367	1196	0.997
Port # 3	195	10176	0	85.0	4.5	205	132	1.000
Total	2832	10010	0	-	12.4	683	2114	0.999
Paramètre de Jain (tous)	-	-	1.0000	0.9991	0.7293	0.7357	0.7337	-
Paramètre de Jain (dernier)	-	-	1.0000	1.0000	0.7639	0.7608	0.5785	-

Tableau 4.19 Statistiques de l'analyse à trafic asymétrique d'un TI 16x16 avec ordonnanceur WRR et un facteur d'accélération de 1.05

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus (1)	Équité de connexion (2)	Occupation des files (VOQ) (3)	Latence moyenne (4)	$99^{ième}$ percentile latence (5)	Taux d'utilisation du lien
Type de ports	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	886	10030	0	94.8	18.0	3003	10968	0.997
Port # 1	869	9888	0	93.5	17.8	3072	8713	0.990
Port # 2	905	10131	0	96.0	16.7	2950	10123	0.982
Port # 3	878	9929	0	94.0	17.6	3156	6278	0.995
Port # 4	880	9981	0	94.4	18.0	3414	9005	0.995
Port # 5	873	9882	0	93.4	18.3	3135	8156	0.993
Port # 6	873	9889	0	93.6	17.7	3177	6750	0.992
Port # 7	868	9808	0	92.8	17.6	3070	7004	0.989
Port # 8	899	10118	0	95.4	16.1	2949	8742	0.998
Port # 9	888	10170	0	96.2	16.6	2840	8055	0.993
Port # 10	877	9892	0	93.5	18.1	3103	6973	0.989
Port # 11	889	10008	0	94.7	17.0	3030	7124	0.986
Port # 12	887	9984	0	94.3	17.0	3062	7185	0.992
Port # 13	869	9793	0	92.9	18.0	3312	7744	0.989
Port # 14	891	10042	0	95.2	17.0	3045	9086	0.992
Port # 15	201	10460	0	98.9	18.4	3258	1149	0.993
Total	13433	10083	0	-	17	3089	9566	0.992
Paramètre de Jain (tous)	-	-	1.0000	0.9998	0.9986	0.998	0.9311	-
Paramètre de Jain (dernier)	-	-	1.0000	0.9994	0.9993	0.9993	0.6386	-

Tableau 4.20 Statistiques de l'analyse à trafic asymétrique d'un TI 16x16 avec ordonnanceur WWFA et un facteur d'accélération de 1.05

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus (1)	Équité de connexion (2)	Occupation des files (VOQ) (3)	Latence moyenne (4)	$99^{ième}$ percentile latence (5)	Taux d'utilisation du lien
Type de ports	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	867	9783	0	93.4	18.3	3236	6525	1.000
Port # 1	869	9907	0	94.5	17.6	3105	7930	1.000
Port # 2	862	9638	0	92.1	17.9	3198	7619	1.000
Port # 3	866	9769	0	93.0	18.0	3215	5836	1.000
Port # 4	864	9799	0	93.4	17.7	3195	5268	1.000
Port # 5	875	9856	0	94.1	18.3	3231	6236	1.000
Port # 6	881	9956	0	95.1	17.9	3165	6080	1.000
Port # 7	883	9971	0	95.1	17.7	3123	5380	1.000
Port # 8	872	9826	0	93.6	16.2	2757	5948	1.000
Port # 9	878	10041	0	95.8	15.6	2869	6928	1.000
Port # 10	900	10130	0	96.3	16.3	2852	5749	1.000
Port # 11	908	10238	0	97.7	16.0	2724	5412	1.000
Port # 12	902	10160	0	97.0	16.4	2979	7911	1.000
Port # 13	909	10250	0	97.6	15.2	2757	5403	1.000
Port # 14	910	10326	0	98.5	14.5	2595	5420	1.000
Port # 15	198	10364	0	98.8	16.2	2623	2008	1.000
Total	13444	10003	0	-	16	2991	7397	1.000
Paramètre de Jain (tous)	-	-	1.0000	0.9996	0.9953	0.9943	0.9515	-
Paramètre de Jain (dernier)	-	-	1.0000	0.9996	0.9995	0.9955	0.7915	-

Tableau 4.21 Statistiques de l'analyse à trafic asymétrique d'un TI 32x32 avec ordonnanceur WRR et un facteur d'accélération de 1.025

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus (1)	Équité de connexion (2)	Occupation des files (VOQ) (3)	Latence moyenne (4)	99ième percentile latence (5)	Taux d'utilisation du lien
Type	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	889	10043	0	98.6	18.3	6282	17153	0.998
Port # 1	877	10028	0	98.2	18.4	5893	16510	0.998
Port # 2	891	10023	0	98.2	18.3	6598	19143	0.993
Port # 3	890	9998	0	97.8	18.7	6842	16619	0.998
Port # 4	889	10066	0	98.6	18.4	6117	16190	0.999
Port # 5	883	9969	0	97.8	18.6	6486	19746	1.000
Port # 6	891	10079	0	98.7	18.4	5967	23246	1.000
Port # 7	882	9965	0	97.6	18.7	5922	17863	0.998
Port # 8	882	9994	0	98.0	19.0	6144	15115	0.996
Port # 9	888	10026	0	98.2	18.4	5952	17651	1.000
Port # 10	879	9953	0	97.7	18.8	6845	17308	0.997
Port # 11	879	10000	0	97.9	18.7	6180	15769	0.998
Port # 12	876	9934	0	97.3	18.8	6221	13648	1.000
Port # 13	880	10016	0	98.2	18.0	6395	14623	0.994
Port # 14	889	10018	0	98.1	18.5	6374	15925	0.997
Port # 15	883	9988	0	97.9	18.0	5860	17586	1.000
Port # 16	878	9999	0	98.1	17.7	6164	16104	0.997
Port # 17	886	9947	0	97.6	18.4	6413	16774	0.999
Port # 18	884	10041	0	98.3	18.4	6180	17247	0.997
Port # 19	883	9941	0	97.4	18.4	6213	16074	0.996
Port # 20	880	10001	0	98.0	18.6	6613	17987	0.997
Port # 21	891	9994	0	97.9	17.9	5774	14777	0.998
Port # 22	880	10019	0	98.2	18.5	6165	18237	0.995
Port # 23	888	9985	0	97.9	18.7	6237	17602	0.995
Port # 24	879	9929	0	97.3	18.3	6811	15849	0.998
Port # 25	870	9911	0	96.9	18.6	6358	17654	1.000
Port # 26	883	9982	0	97.8	18.6	5999	15955	0.997
Port # 27	883	9995	0	97.8	18.2	6033	19867	0.999
Port # 28	892	10098	0	99.0	18.0	6502	19268	0.998
Port # 29	891	10038	0	98.2	18.6	6368	15453	0.999
Port # 30	884	9921	0	97.2	18.6	6374	16740	0.996
Port # 31	196	10108	0	98.5	20.0	6454	4437	0.995
Total	27596	10025	0	-	18	6269	18237	0.998
Jain 1	-	-	1.0000	1.0000	0.9996	0.998	0.9717	-
Jain 2	-	-	1.0000	1.0000	0.9983	0.9998	0.7433	-

Tableau 4.22 Statistiques de l'analyse à trafic asymétrique d'un TI 32x32 avec ordonnanceur WWFA et un facteur d'accélération de 1.025

Statistique	Nombre de paquets analysés	Nombre de cycles analysés	Nombre de paquets perdus (1)	Équité de connexion (2)	Occupation des files (VOQ) (3)	Latence moyenne (4)	$99^{ième}$ percentile latence (5)	Taux d'utilisation du lien
Type	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Entrée	Sortie
Unité	Paquets	Cycles	Paquets	%	%	Cycles	Cycles	-
Port # 0	856	9712	0	95.7	19.5	6705	13839	1.000
Port # 1	866	9837	0	96.6	18.6	6441	11444	1.000
Port # 2	877	9872	0	97.1	19.0	6592	13031	1.000
Port # 3	883	9967	0	97.8	18.7	6423	11957	1.000
Port # 4	871	9851	0	96.7	18.0	6262	9732	1.000
Port # 5	884	10033	0	98.6	18.6	6343	11309	1.000
Port # 6	855	9732	0	95.6	18.9	6907	10900	1.000
Port # 7	883	9903	0	97.2	18.8	6641	12827	1.000
Port # 8	872	9826	0	96.5	19.3	6613	11430	1.000
Port # 9	872	9833	0	96.7	18.1	6355	11895	1.000
Port # 10	871	9889	0	97.1	18.4	6464	10414	1.000
Port # 11	875	9915	0	97.2	18.7	6781	12022	1.000
Port # 12	888	9995	0	98.2	18.9	6451	11756	1.000
Port # 13	878	10003	0	98.4	18.4	6467	10708	1.000
Port # 14	894	10061	0	98.9	18.4	6561	11693	1.000
Port # 15	896	10140	0	99.6	17.8	6335	11636	1.000
Port # 16	879	10001	0	98.2	17.6	6279	10510	1.000
Port # 17	880	9966	0	97.8	18.1	6489	10862	1.000
Port # 18	882	10007	0	98.3	19.1	6400	10390	1.000
Port # 19	879	9930	0	97.7	18.7	6665	10546	1.000
Port # 20	866	9838	0	96.6	18.4	6336	9632	1.000
Port # 21	894	10012	0	98.4	17.8	6068	11914	1.000
Port # 22	888	10135	0	99.7	17.5	6136	11523	1.000
Port # 23	891	10000	0	98.2	17.6	6304	11039	1.000
Port # 24	902	10227	0	100.4	16.8	5996	12009	1.000
Port # 25	900	10262	0	100.7	17.2	5959	10479	1.000
Port # 26	895	10107	0	99.3	17.3	6153	10396	1.000
Port # 27	891	10122	0	99.3	16.9	6003	11898	1.000
Port # 28	897	10145	0	99.7	17.3	6284	12457	1.000
Port # 29	910	10266	0	100.8	16.9	5917	11201	1.000
Port # 30	913	10319	0	101.4	16.2	5725	11466	1.000
Port # 31	198	10107	0	98.8	19.2	5918	4830	1.000
Total	27586	10001	0	-	18	6350	13289	1.000
Jain 1	-	-	1.0000	0.9998	0.998	0.9981	0.9833	-
Jain 2	-	-	1.0000	1.0000	0.9992	0.9987	0.8595	-

Les trois histogrammes des figures suivantes permettent de comparer visuellement les paramètres de Jain (1) entre tous les ports d'entrée des diverses statistiques d'équité pour les trois dimensions de TI simulées. Les étiquettes 1 à 5 de l'axe des X représentent les paramètres identifiés aux tableaux précédents avec ces mêmes étiquettes.

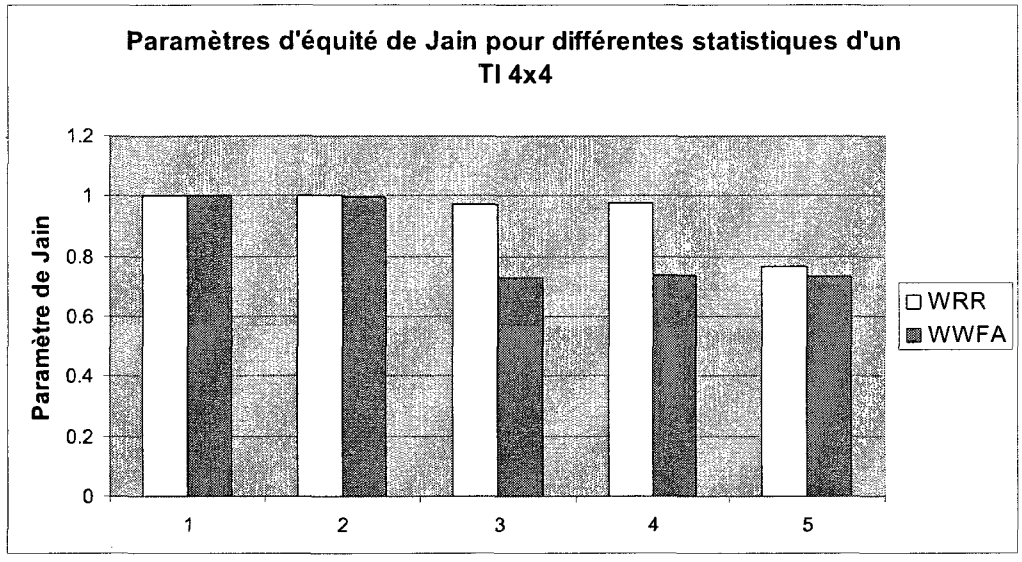

Figure 4.2 Comparaison des paramètres de Jain pour différentes statistiques pour l'ordonnanceur WRR et WWFA d'un TI 4x4

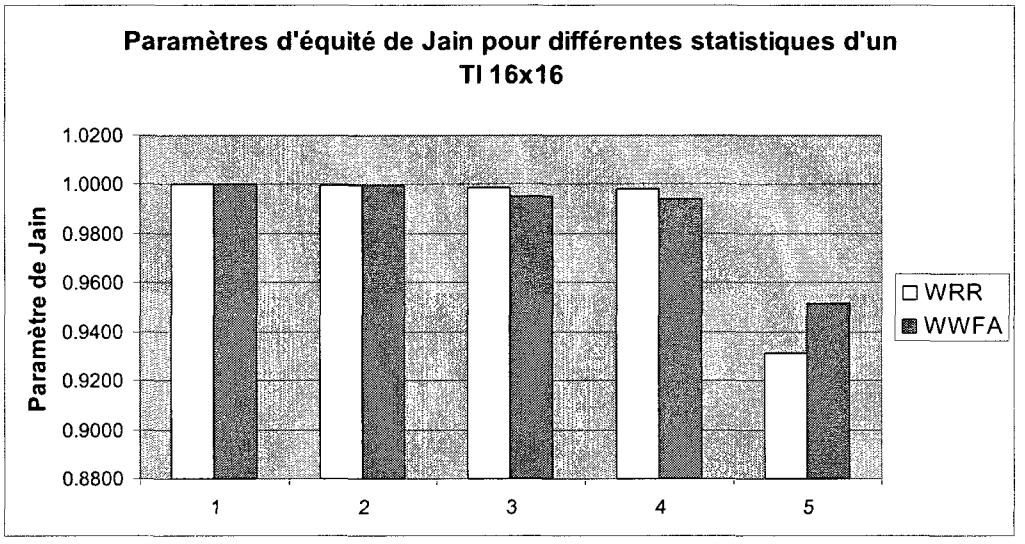

Figure 4.3 Comparaison des paramètres de Jain pour différentes statistiques pour l'ordonnanceur WRR et WWFA d'un TI 16x16

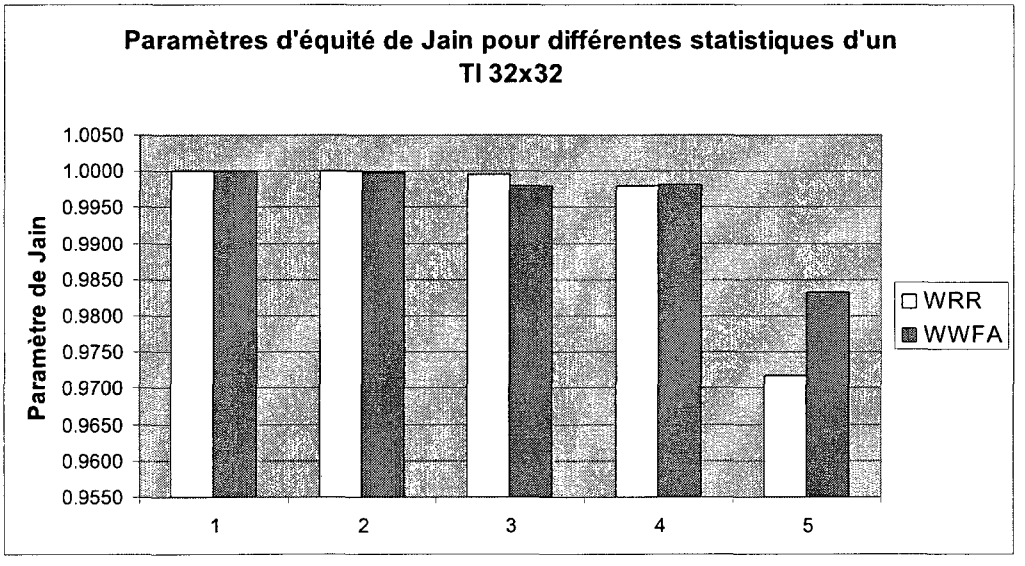

Figure 4.4 Comparaison des paramètres de Jain pour différentes statistiques pour l'ordonnanceur WRR et WWFA d'un TI 32x32

D'après les paramètres de Jain des histogrammes, on observe que le WRR semble plus équitable, mais cette conclusion n'est pas générale. Cette conclusion est le plus aisément observable pour le TI 4x4. L'occupation des VOQ par port et la latence moyenne par port sont particulièrement débalancées pour l'entrée 3 par rapport aux autres entrées. Les paramètres de Jain pour ces statistiques montrent également ce débalancement, comme on peut le voir pour les paramètres (3) et (4) de l'histogramme présenté à la figure 4.2. Les équités de connexion sont égales pour les deux algorithmes. Le nombre de paquets perdus n'est pas pertinent ici puisque les VOQ sont intentionnellement très gros afin de ne pas influencer les résultats d'équité. Le paramètre de Jain du 99^{ième} percentile semble indiquer une équité meilleure de 3% pour l'algorithme WRR, ce qui est une différence relativement faible. Par contre, la valeur elle-même du 99^{ième} percentile est également une mesure de l'équité. Pour ce paramètre, le WWFA est le plus équitable. Donc, bien que le TI 4x4 donne les résultats les plus drastiques, il n'est pas évident de trancher à savoir quel algorithme est le plus équitable. On peut affirmer que les deux ordonnanceurs sont aussi équitables l'un que l'autre pour un TI 4x4 lorsqu'on simule avec un trafic asymétrique.

Pour les TI de 16 et 32 ports, l'analyse est plus complexe. Puisqu'il n'y a qu'un seul port avec des paquets de taille plus grosse, l'impact de ce port sur le paramètre de Jain pour tous les ports devient moins significatif. Donc, l'effet du dernier port sur ces paramètres de Jain est dilué. Par contre, le paramètre de Jain entre le dernier port et la moyenne des autres ports devient intéressant et plus aisé à observer. On observe, avec le paramètre de Jain, que la latence et l'occupation des VOQ semble légèrement plus équitable pour le WRR. Par contre, le paramètre de Jain pour le 99^{ième} percentile semble montrer l'opposé. La différence est encore plus significative pour le deuxième paramètre de Jain de cette même statistique. Encore une fois, la valeur du 99^{ième} percentile indique que le WWFA est significativement plus équitable.

Pour le TI à 32 ports, l'analyse est relativement similaire à celle du TI à 16 ports. La différence la plus significative par rapport à l'analyse avec un TI de 16 ports relative au paramètre de Jain est observée pour la latence. Celle-ci est similaire pour les 2 algorithmes d'ordonnancement avec un TI de 32 ports. La valeur du 99^{ième} percentile indique encore que le WWFA est plus équitable.

En conclusion, on a pu observer différentes statistiques et en conclure les différences de comportement des deux algorithmes lorsqu'il y a un trafic asymétrique. Cependant, on peut conclure, de façon générale, que les deux algorithmes sont aussi équitables l'un que l'autre pour les trois tailles de TI simulées. Donc, puisque que l'algorithme WRR n'est pas réalisable en matériel avec un haut débit à cause de la complexité de sa logique et de la longueur de ses fils, on a prouvé que le WWFA est potentiellement intéressant car ce dernier permet une implémentation plus aisée tout en ne sacrifiant pas l'équité.

Il est à noter que l'analyse d'équité est celle obtenue alors que les files VOQ sont d'une profondeur de 100 paquets, ce qui est peu réaliste dans un système réel. Il serait intéressant d'intégrer à l'analyse d'équité un paramètre supplémentaire : la taille de ces files.

En analysant davantage la latence moyenne, le 99^{ième} percentile de la latence et le taux d'utilisation des ports de sortie, on remarque que, pour toutes les simulations, l'algorithme WWFA est plus performant, bien qu'aussi équitable que le WRR. On observe également que le facteur d'accélération choisi pour un trafic symétrique s'avère légèrement insuffisant pour les TI de 16 et 32 ports avec l'algorithme d'ordonnancement WRR lorsqu'on a un trafic asymétrique. En effet, le taux d'utilisation des liens de sortie est inférieur à 1.0 pour ces simulations. Pour le TI à 4 ports qui utilise l'algorithme WRR, la différence est encore plus significative avec un taux d'utilisation de seulement 0.9171.

Un facteur d'accélération supérieur aurait amélioré les performances pour ces simulations, surtout pour le TI à 4 ports.

Cette section a permis de montrer comment l'environnement fournit les outils nécessaires à la détermination de l'équité. De plus, le résultat de la comparaison de l'équité du WRR et du WWFA montre qu'ils sont pratiquement aussi équitables l'un que l'autre, ce qui constitue un point en faveur de l'utilisation du WWFA puisqu'il est plus facile à implémenter en matériel. La prochaine section montre un exemple de caractérisation d'un TI.

4.2.4 Caractérisation d'un TI en fonction du trafic d'entrée

Lors du choix d'une architecture de TI, on s'intéresse habituellement à son comportement pour divers trafics d'entrée. Il est possible de produire des graphiques montrant des paramètres divers en fonction du trafic d'entrée. Cela permet de caractériser le comportement d'un TI. Ceci est réalisé dans cette section. En l'occurrence, on s'intéresse à la latence moyenne et au $99^{ième}$ percentile de la latence, encore une fois, pour les deux algorithmes d'ordonnancement et les trois tailles de TI. Il est pertinent d'identifier le trafic maximal pour lequel ces statistiques commencent à se détériorer de façon significative.

Un trafic symétrique est utilisé. Une taille de 50 à 90 octets est conservée pour les entrées. Le trafic en entrée est fixé à 25, 50, 75, 90 et 100 % de la bande passante maximale. Le facteur d'accélération utilisé est fixé à celui prouvé comme satisfaisant dans une section précédente. La destination est, encore une fois, aléatoire pour toutes les entrées. La taille des VOQ internes est fixée à 10 paquets. Contrairement aux autres simulations réalisées, on ne laisse pas le temps aux files de se remplir en début de simulation puisque, en régime permanent, les files ne devraient pas avoir un grand taux d'occupation pour un trafic qui n'utilise la bande passante qu'offre les liens que

partiellement. L'analyse statistique commence après *(150* nombre de ports du TI)* cycles de simulation (total), permettant de s'assurer d'être en régime permanent. L'analyse se fait pour, en moyenne, 10 000 cycles par port. On s'assure évidemment que la simulation est toujours en régime permanent au moment de terminer l'analyse. Les tableaux 4.23 à 4.25 détaillent les paramètres de configuration pour le TI, la génération du trafic aléatoire et l'analyse statistique.

Tableau 4.23 Configuration du TI utilisé pour l'évaluation d'équité

Champ	Valeur
Type d'ordonnanceur	WRR vs WWFA
Nombre de ports du TI	4, 16 et 32
***Facteur d'accélération du commutateur* crossbar**	1.2 pour 4 ports; 1.05 pour 16 ports; 1.025 pour 32 ports
TI SystemC vs RTL	TI SystemC
Poids par port pour l'ordonnanceur WRR	1
Profondeur des files	10 paquets
Profondeur des FIFO de sortie	2 paquets
Largeur des ports	56 bits
Période d'horloge	4 ns
Nombre de cycles d'initialisation	0 cycle
Nombre de paquet moyen par file VOQ (initialisation)	0 paquets

Tableau 4.24 Paramètres de la génération du trafic pour l'évaluation d'équité

Nom du paramètre	valeur/ unité	Valeur
NB_PACKETS	Nb. paquets	10 000
RANDOM_DEST	on/off	On
BW_MIN_DL	Nb. octets	50
BW_MAX_DL	Nb. octets	90
BW_PERC_BW	0-100 %	100%
MUL_FACTOR_BW_PERC_BW [NB_PORT]	Ratio	1
MUL_FACTOR_BW_DL [NB_PORT]	Ratio	1
N_CONSEC	Entier	2
SCV	Interrupteur	On
NB_PACKET_SAME_DEST	Entier	1

Tableau 4.25 Paramètres de l'analyse statistique pour l'évaluation d'équité

Nom du paramètre	Valeur
$NB_INIT_CYCLES	*150* nombre de ports du TI*
$NB_TOT_DATACYCL_ANAL	10 000 par port
$NB_CYCLE_MAX_IDLE	200

Le tableau 4.26 montre les statistiques de latence et de 99^{ième} percentile de la latence obtenues suite aux simulations. L'ensemble des résultats est présenté pour les deux ordonnanceurs ainsi que pour un TI de 4, 16 et 32 ports.

Tableau 4.26 Latence et 99^{ième} percentile de la latence

Taille	Algorithme d'ordonnancement	Trafic	Latence moyenne	99^{ieme} percentile de la latence
4 ports	WRR	0.25	13	29
		0.5	16	46
		0.75	24	80
		0.9	40	147
		1	117	394
	WWFA	0.25	13	29
		0.5	16	49
		0.75	24	76
		0.9	40	157
		1	117	426
16 ports	WRR	0.25	14	33
		0.5	17	57
		0.75	28	117
		0.9	56	265
		1	164	884
	WWFA	0.25	14	33
		0.5	17	53
		0.75	28	112
		0.9	56	282
		1	164	882
32 ports	WRR	0.25	20	71
		0.5	29	151
		0.75	65	444
		0.9	138	763
		1	361	1844
	WWFA	0.25	21	66
		0.5	29	132
		0.75	65	401
		0.9	138	806
		1	359	1812

Les graphiques des figures 4.5 et 4.6 montrent les 2 paramètres en fonction du trafic en entrée.

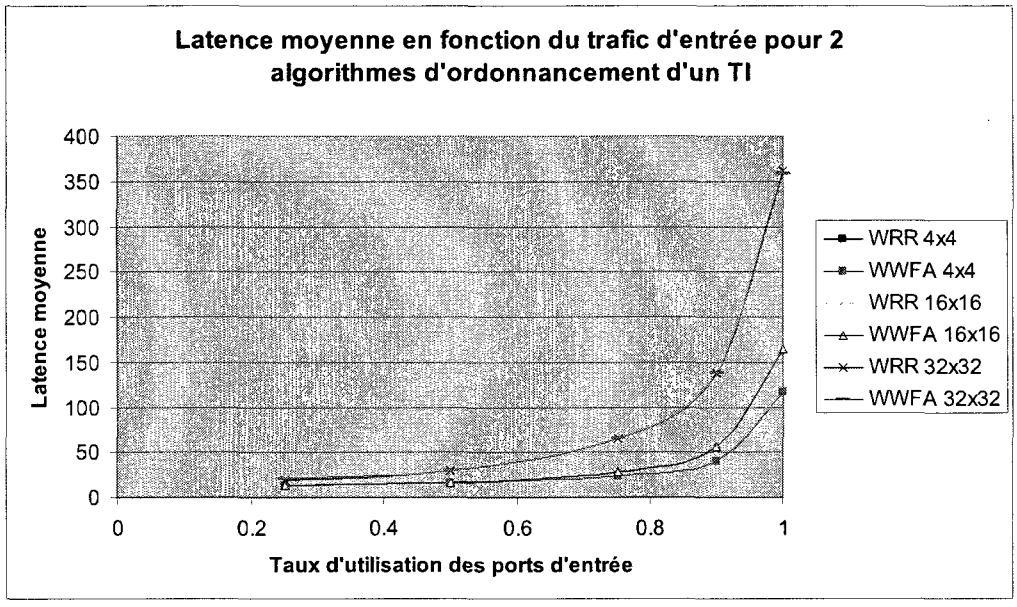

Figure 4.5 Latence en fonction du trafic d'entrée

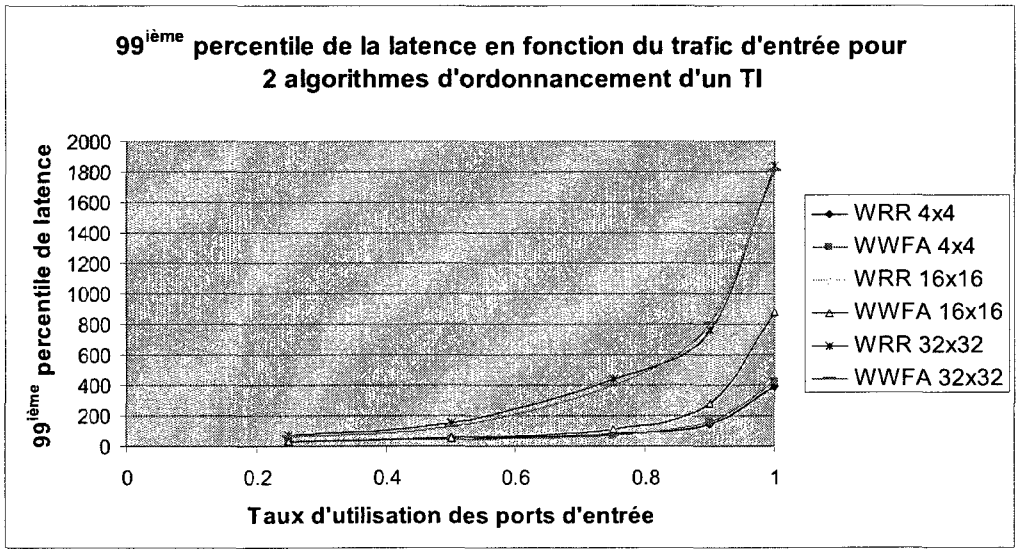

Figure 4.6 99^{ième} percentile de la latence en fonction du trafic d'entrée

Le 99^{ième} percentile de la latence est un indicatif de l'équité du TI. En ce sens, les courbes de ce paramètre permettent de connaître l'équité du TI utilisé pour différents trafics. On remarque que le type d'ordonnanceur a très peu d'influence pour les scénarios testés. On remarque également que, plus le nombre de ports du TI est élevé, plus les deux paramètres analysés se dégradent pour un trafic en entrée se rapprochant de 100%. Également, on remarque que les TI à 4 et 16 ports ont des statistiques relativement constantes et semblables pour un trafic inférieur à environ 0.75. Par contre, le TI avec 32 ports est moins performant avec un trafic de 0.25 à 0.75. On observe également que la performance obtenue n'est pas constante pour un trafic inférieur à 0.75 comme il est le cas pour les TI à 4 et 16 ports. Les paramètres commencent déjà à montrer une performance moindre au fur et à mesure que le trafic augmente, et ce, même si le trafic demeure inférieur à 0.75. Ces courbes permettent d'établir les performances attendues dans un scénario spécifique de trafic en entrée et de potentiellement accepter/rejeter un TI spécifique pour une application particulière.

4.3 Conclusion

Bien que le but principal du projet ne soit pas de faire des études statistiques sur des architectures, mais bien de réaliser un environnement de validation de performance et de présenter une méthode de validation/vérification efficace qui permet d'intégrer la validation de performance au processus de conception, ce chapitre montre des exemples réels d'utilisation de l'environnement et les conclusions qui en découlent. Il a été démontré qu'il est aisé, avec l'environnement créé, de modéliser, de simuler et de combiner les résultats pour tirer les conclusions appropriées. Les TI ont étés utilisés comme exemple de circuits avec une architecture à définir mais les mêmes techniques et principes s'appliquent pour une large classe de circuits intégrés. Ce chapitre montre aussi les étapes de la méthode de conception efficace utilisée et suggérée. Il existe beaucoup d'autres études possibles sur l'architecture d'un TI qui pourraient être faites dans le cadre d'un projet ultérieur avec l'environnement réalisé.

CONCLUSION

Ce mémoire a permis principalement d'explorer l'aspect de validation de performance rencontrée dans le processus de conception d'un circuit intégré. L'approche prise a été d'utiliser un langage connu pour la vérification avancée comme levier pour valider la performance. Cela a permis d'obtenir la contribution principale de ces travaux, c'est-à-dire, un environnement commun pour la vérification et la validation de performance. Il découle également des travaux une approche efficace qui s'intègre au processus de conception de circuits intégrés. Les tissus d'interconnexion étant des circuits à complexité sans cesse croissante, ils ont servi d'architecture à valider et vérifier pour ces travaux.

La première section de ce mémoire montre les divers aspects théoriques reliés au projet. On y définit la validation de performance, la vérification et on y présente des bibliothèques C++ modernes de modélisation/vérification : SystemC et SCV. On y présente aussi des éléments théoriques propres aux TI nécessaires à la compréhension du mémoire. Le deuxième chapitre présente l'environnement de validation de performance et vérification à haut niveau, de même que les TI matériels et à haut niveau d'abstraction réalisés. La première partie du chapitre 3 amène un aspect d'application réelle en intégrant un processeur configurable à l'environnement. Le système réalisé est pour une application vidéo, en l'occurrence, la compensation de mouvement. La section suivante de ce même chapitre propose la méthode à utiliser afin d'intégrer un modèle de trafic synthétique disponible à l'environnement, augmentant ainsi le réalisme de la génération de trafic pour une application spécifique. Cela permet également d'intégrer des modèles de trafic C/C++ existants dans l'environnement, évitant de recommencer du début l'implémentation de modèles de trafic pertinents. Ces trois aspects de l'environnement couverts aux chapitres 2 et 3 sont complémentaires et contribuent individuellement à former une plateforme complète et efficace de validation/vérification. Le but du projet

n'était pas de démontrer l'efficacité d'une architecture particulière, mais bien d'explorer la validation de performance avec un environnement de simulation et de déterminer une méthode pour l'intégrer au processus de conception de TI. Cette méthode est résumée au début du 4^{e} chapitre. Ce dernier chapitre montre également des exemples de résultats de simulations obtenus avec l'environnement, ce qui permet de bien constater la pertinence de l'environnement. Dans ce chapitre, le facteur d'accélération d'un TI nécessaire pour un trafic élevé mais symétrique a été déterminé. Il a été montré qu'un facteur d'accélération significatif donne peu de bénéfices pour un tissu avec un grand nombre de ports. Également, l'équité de deux ordonnanceurs lorsqu'il y a un trafic asymétrique a été évaluée. En plus d'être plus propice à une implémentation matérielle, on arrive à la conclusion que l'ordonnanceur WWFA étudié conserve une équité similaire au WRR. Enfin, on a pu caractériser des TI en fonction du trafic en entrée en analysant la latence moyenne et le 99^{ième} percentile de la latence des paquets. Ce dernier paramètre sert de paramètre d'équité. Bien entendu, beaucoup d'autres analyses sont possibles avec l'environnement créé.

On peut donc conclure que l'ensemble des réalisations du projet forme un environnement complet qui touche toutes les phases de la conception d'un TI. Comme projet ultérieur, il y aurait la poursuite des travaux d'exploration architecturale pour les TI en y intégrant des nouvelles idées architecturales, la création d'un environnement avec un processeur Tensilica plus complexe et efficace pour la compensation de mouvement, et enfin, appliquer les concepts présentés afin d'intégrer un modèle de trafic existant à l'environnement.

RÉFÉRENCES

[1] ARMSTRONG J.R., GRAY F.G>, et VUPPALA S., "Methodology for VHDL Performance Model Construction and Validation", Proceedings IEEE Southeastcon '97 "Engineering new New Century", Avril 1997, pp. 29-35.

[2] BALAKRISHNAN M., JAIN M. K., et KUMAR A., "ASIP design methodolgies:Survey and issues", Proceedings of IEEE / ACM International Conference on VLSI Design, 2001, pp. 76-81.

[3] BÉLANGER N., LEBEL D., MBAYE M., PIERRE S., et SAVARIA Y., "Design exploration with an application-specific instruction-set processor for ela dein-terlacing,". Proceedings 2006 IEEE International Symposium on Circuits and Systems (ISCAS), 2006.

[4] BERGERON J., "Writing Testbenches, Functional Verification of HDL Models", 2nd edition, Springer, 2002.

[5] BEUCHER N., « Conception et Mise en Oeuvre de Processeurs Configurables Pour la Conversion de Taux de Trames Vidéos avec Compensation de Mouvement », M.Sc.A., École Polytechnique de Montréal, Qc, Canada, Juillet 2007.

[6] BRAHME D. S., et al, "The Transaction-Based Verification Methodology", Cadence Berkeley Labs, Berkeley, California, USA, Rapport Technique # CDNL-TR-2000-0825, Août 2000.

[7] CADENCE, "Incisive Plan-to-Closure methodology, an automated plan- and metric-driven approach to system-level verification closure". [En ligne]. http://www.cadence.com/products/fv/plan_to_closure/pages/default.aspx.

[8] CADENCE, "White paper on an unified verification methodology". [En ligne]. http://www.cadence.com/rl/Resources/white_papers/4442_Unified_VerificationMethodology_WP1.pdf.

[9] CALAZANS E. M., CARARA E., HESSEL F., MORAES F., MORENO E., et ROSA V., "From VHDL Register Transfer Level to SystemC Transaction Level Modeling: a Comparative Case Study", Proceedings Symposium on Integrated Circuit and Systems Design, 2003, pp. 355-360.

[10] CASTAGNO R., HAAVISTO P., et RAMPONI G., "A method for motion adaptive frame rate up-conversion,". Circuits and Systems for Video Technology, IEEE Transactions on 6 (1996), pp. 436–446.

[11] DELGADO-FRIAS J. G., et RATNPAL G. B., "A VLSI Crossbar Switch with Wrapped Wave Front Arbitration", IEEE transationc on circuits and systems, vol 50, no 1, Janvier 2003.

[12] DRUCKER L., KHAN N., et SINGH L., "Advanced Verification Techniques: A SystemC Based Approach for Successful Tapeout", Kluwer Academic Publishers, 2004.

[13] GLASSER M., ROSE A., FITZPATRICK T., RICH D., et FOSTER H., "Advance verification methodology cookbook" (AVM), Mentor Graphics, version 2.0, Janvier 2004.

[14] GNUPLOT. Description de l'outil "gnuplot". [En ligne].
http://www.gnuplot.info/.

[15] HURT J., LIN B., MAY A., et ZHU X., "Design and Implementation of High-speed Symmetric Crossbar Schedulers", Proceedings IEEE International Conference Communications (ICC), Juin 1999, pp. 1478-1483.

[16] IEEE 1666 LRM. [En ligne].
http://www.systemc.org/downloads/lrm.
http://standards.ieee.org/getieee/1666/index.html.

[17] JAIN R., CHIU D., et HAWE W., "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems", Digital Equipment Corporation, Maynard, MA, USA, Dec Research Report TR-301, September 1984.

[18] KEUTZER K., MALIK S., et NEWTON R., "From ASIC to ASIP: the next design discontinuity", Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD), Janvier 2002.

[19] LIU Y., TIAN Y., TU X., WANG K., et WEN L., "PPAwFE: A novel high speed crossbar scheduling algorithm", Proceedings International Conference on Communications, Circuits and Systems, 2005, pp. 673-677.

[20] LOISEAU L., Méthodologie design-reuse, École Polytechnique de Montréal, Département de génie électrique, Miranda Technoloigies Inc., 2001

[21] MANDVIWALLA M., et TZENG N-F., "Cost-Effective Switching Fabrics with Distributed Control for Scalable Routers", Proceedings 22nd International Conference on Distributed Computing Systems, 2002, pp. 65-73.

[22] MARTINELLI P., WELLIG A., et ZORY J., "Transaction-level prototyping of a UMTS outer-modem for System-on-chip validation and architecture exploration", Proceedings IEEE International Workshop on Rapid System Prototyping, 2004, pp. 193-200.

[23] MENTOR GRAPHICS INC., "Modelsim Advanced Verification and Debugging", SE User's Manual, version 6.0c, 21 Janvier 2005.

[24] NORLUND K., OTTOSSON T., et BRUNSTORM A., "TCP fairness measure for scheduling algorithms in wireless networks", Proceedings of the 2nd Int'l Conf. on Quality of Service in Heterogeneous Wired/Wireless Networks (QShine'05), 2005.

[25] OPEN VERIFICATION METHODOLOGY. [En ligne].
http://www.ovmworld.org.

[26] PARK S., et CHAE S., "A C/C++-based Functional Verification Framework using the SystemC Verification Library", Rapid System Prototyping (RSP), Juin 2005, pp.237-239.

[27] RASHINKAR P., et al, "SOC Verification, Methodology and Techniques", Kluwer Academic Publishers, fourth printing, 2003.

[28] SEIFERT R., "The Switch Book: The Complete Guide to LAN Switching Technology", John Wiley & Sons, Inc., 2000.

[29] SPEAR C., "SystemVerilog for Verification: a Guide to Learning the Testbench Language Features", 2nd edition, Springer, 2008.

[30] SYSTEMC ORGANIZATION. [En ligne].
http://www.systemc.org.

[31] SYSTEMC ORGANIZATION, Standards des bibliothèques SystemC & SCV. [En ligne].
http://www.systemc.org/downloads/standards.

[32] TAMIR Y., et CHI H., "Decomposed Arbiters for Large Crossbars with Multi-Queue Input Buffers", Proceedings of the International Conference on Compter Design, Octobre 1991, pp.233-238.

[33] TAMIR Y., et CHI H., "Symmetric Crossbar Arbiters for VLSI Communication Switches", IEEE transactions on parallel and distributed systems, vol. 4, no. 1, 1993.

[34] TAMIR Y., et FRAZIER G., "High performance multi-queue buffers for VLSI communication switches," Proceedings 15th Annu. Symp. Comput. Arch., Juin 1988, pp. 343–354.

[35] TENSILICA INC., "Xtensa SystemC (XTSC) User's Guide", 2006.

[36] TUNDRA SEMICONDUCTOR. [En ligne].
http://www.tundra.com.

[37] TUTSCH D., "Performance Analysis of Network Architectures", Springer, 2006.

[38] WIKIPEDIA, Description de la compensation de mouvement. [En ligne].
http://en.wikipedia.org/wiki/Motion_compensation.

[39] WIKIPEDIA, Description des réseaux "fully connected". [En ligne].
http://en.wikipedia.org/wiki/Fully-connected_network.

[40] WIKIPEDIA, Description du "Weighted Round-robin". [En ligne].
http://en.wikipedia.org/wiki/Weighted_round_robin.

[41] ZHENG H., ZHAO Y., et CHEN C., "Design and Implementation of Switches in Network Simulator (*ns2*)", Proceedings of the First International Conference on Innovative Computing, Information and Control (ICICIC'06), 2006.

CONTENU DES ANNEXES

ANNEXE A. IMPLÉMENTATION DU TI MATÉRIEL

L'architecture sommaire du TI matériel réalisé est présentée au chapitre 2. Cette annexe est un complément qui donne une vue plus détaillée du TI matériel. Entre autres, les aspects d'implémentation et les ports des différents sous-modules réalisés sont présentés et décrits ici.

A.1 Module du VOQ (« Virtual Output Queues »)

La figure a.1 illustre les ports du module de « Virtual Output Queues » et les principales opérations qu'il effectue. Ce module emmagasine les données reçues à l'entrée dans des FIFO en fonction de la destination. Lorsqu'un des FIFO reçoit une donnée alors qu'il était vide lors du cycle précédent, le signal *« do_schedule »* s'active. Cela indique à l'ordonnanceur qu'une donnée a changé et qu'une nouvelle planification s'avère nécessaire. À chacune des lectures d'un paquet d'un des FIFO, le module est notifié afin qu'il mette à jour le nombre de paquets contenu dans chacun des FIFO. Dès qu'on commence à lire un nouveau paquet, on doit spécifier le signal *« start_of_packet_pop »*. Ce signal sert à mettre à jour le compteur de paquets à l'intérieur du module VOQ. Ceci assure également qu'un paquet déjà planifié ne sera pas planifié deux fois par l'ordonnanceur. Le module présent fournit l'information sur la longueur du paquet courant en termes de mots (valeur synchronisée avec la donnée lue). Cette valeur (*« voq_length »*) est gardée constante pour toute la lecture du paquet. Le module fournit également le nombre de paquets disponibles dans chacun des FIFO sous forme d'une table. Si le module reçoit un paquet pour lequel il n'a pas l'espace mémoire nécessaire (dans le FIFO correspondant), celui-ci est mis de côté au complet sans aucune autre notification.

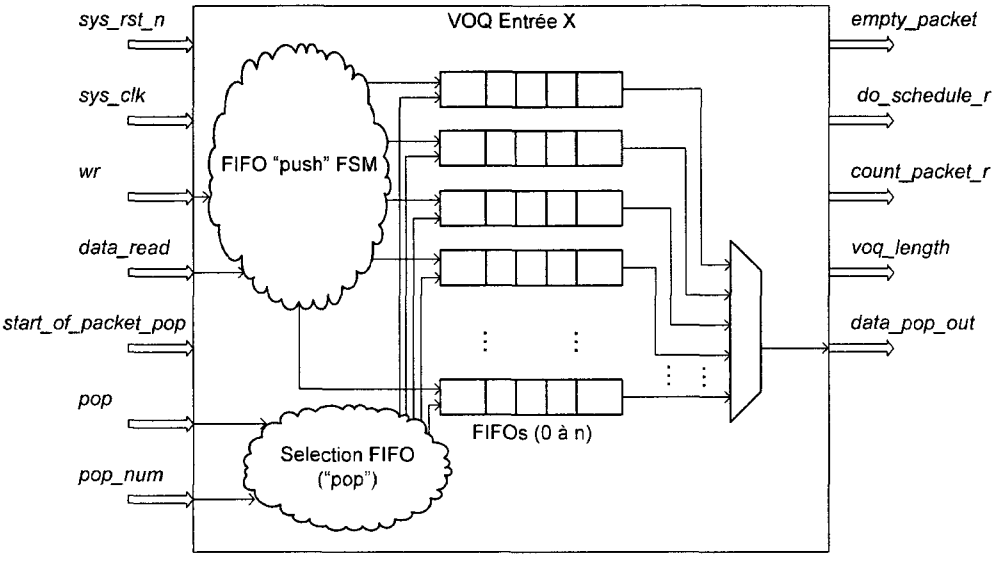

Figure A.1 Schéma bloc du VOQ

Les tableaux a.1 et a.2 décrivent les entrées/sorties du module.

Tableau A.1 Liste des paramètres génériques du VOQ

Générique	Type	Description
NB_PORT	Entier	Nombre de ports de sortie
NB_PORT_LOG2	Entier	Log_2 de NB_PORT (arrondi vers le haut)
NB_BITS	Entier	Taille du mot d'entrée/sortie pour chacun des ports
NB_BITS_ADDR	Entier	Indique la taille du champ qui contient l'adresse de source /destination et est incluse dans l'entête du paquet
NB_BITS_PACKET_ID	Entier	Indique la taille d'un champ définissant l'ID du paquet
NB_BITS_PAYL	Entier	Indique la taille d'un champ définissant le « payload » du paquet (pas utilisé pour ce projet)
NB_BITS_DATA_L	Entier	Indique la taille d'un champ définissant la taille des données contenues dans le paquet en octets mais en n'incluant pas la taille de l'entête du paquet
FIFO_DEPTH	Entier	La profondeur des FIFO du module VOQ

Tableau A.2 Description des ports du VOQ

Champ	Dir.	Taille (bits)	Description	Connectivité
sys_clk	in	1	L'horloge	-
sys_reset_n	in	1	Reset asynchrone actif bas	-
wr	in	1	Force l'écriture au FIFO le mot en entrée	Entrée du TI
data_read	in	NB_BITS	Mot en entrée du TI/VOQ	Entrée du TI
start_of_packet_pop	in	NB_PORT	Il indique qu'on commence la lecture d'un nouveau paquet (dans la FIFO spécifiée)	send_pkt_ctrl
pop	in	1	Il indique qu'on veut effectuer une lecture dans un FIFO.	send_pkt_ctrl (voq_sel)
pop_num	in	NB_PORT_LOG2	Il précise le mot à lire du FIFO correspondant	send_pkt_ctrl (voq_ctrl)
Empty	out	1	Indique que le FIFO correspondant est vide	Ordonnanceur, send_pkt_ctrl,
data_pop_out	out	NB_BITS	Présente la sortie retirée d'un FIFO	commutateur crossbar
voq_length	out	NB_BITS_DATA_L *(en théorie : round_up(log2(8*2 NB_BITS_DATA_L + HEADER_LENGTH)/N B_BITS))*	Nombre de mots total à retirer du FIFO pour le paquet courant (synchronisé avec le data_pop_out)	send_pkt_ctrl
do_schedule_r	out	1	Indique que l'ordonnanceur doit faire une nouvelle planification	Ordonnanceur
count_packet_r	out	Tableau d'entier avec NB_PORT éléments	Indique le nombre de paquets dans chaque FIFO	Ordonnanceur

A.2 Ordonnanceur

Les tableaux a.3 et a.4 décrivent respectivement les paramètres génériques du module ordonnanceur et de ses ports.

Tableau A.3 Description des paramètres génériques de l'ordonnanceur

Générique	Type	Description
NB_PORT	Entier	Nombre de ports de sortie
NB_PORT_LOG2	Entier	Log_2 de NB_PORT (arrondie vers le haut)
NB_BITS	Entier	Taille du mot d'entrée
NB_BITS_ADDR	Entier	Indique la taille du champ qui contient l'adresse de source/destination et qui est incluse dans l'entête du paquet

Tableau A.4 Description des ports de l'ordonnanceur

Champ	Dir.	Taille (bits)	Description	Connectivité
sys_clk	in	1	L'horloge	-
sys_reset_n	in	1	Reset asynchrone actif bas	-
do_schedule_voq	in	1	Demande d'ordonnancement du VOQ	VOQ
do_schedule_outpfreed	in	1	Demande d'ordonnancement du module d'envoi de paquets	send_pkt_ctrl
outp_freed	in	NB_PORT	Indique quelle sortie est libérée et doit être re-planifiée	send_pkt_ctrl
voq_empty	in	NB_PORT	Indique si le FIFO courant est vide	send_pkt_ctrl, VOQ

Champ	Dir.	Taille (bits)	Description	Connectivité
voq_nb_packet	in	Table 2D de NB_PORT x NB_PORT entiers	Indique le nombre de paquets dans chacun des FIFO des modules de VOQ	VOQ
outp_read	in	NB_PORT	Indique que la sortie du TI fait une lecture	Entrée du TI, send_pkt_ctrl
scheduled_outp_r	buffer	Table d'entiers	Table de décision de planification de port d'entrée par sortie	send_pkt_ctrl,
scheduled_outp_nb_packet_r	buffer	Table d'entiers	Nombre de paquet planifiés pour la connexion courante	send_pkt_ctrl,
do_schedule_voq_done_r	out	1	Indique une nouvelle planification suite à une requête de VOQ	send_pkt_ctrl
do_schedule_outpfreed_done_r	out	1	Indique une nouvelle planification suite à une sortie libérée	send_pkt_ctrl,
do_schedule_new_outpread_done_r	out	1	Indique une nouvelle planification suite à une nouvelle lecture de paquet en sortie du TI	send_pkt_ctrl
do_schedule_port_done_r	out	NB_PORT	Indique quels ports ont étés re-planifiés	send_pkt_ctrl

Toutes les sorties de ce module sont registrées. Ce module exécute une boucle pour permettre l'analyse de toutes les connexions entrée/sortie possibles en un seul cycle. Il s'agit d'un algorithme de « weighted round-robin ». La priorité varie à chaque planification de manière à donner la priorité à une sortie différente à chaque fois qu'on fait une planification. Une fois toutes les sorties testées avec une priorité prédominante, on refait la boucle en appliquant un autre « round-robin » sur le port d'entrée, soit sur les

FIFO à l'intérieur des VOQ. Par exemple, pour un TI de 4 ports, il y a au total 16 possibilités pour le choix de la plus haute priorité. À chaque planification, on analyse les 16 possibilités de connexion entrée/sortie en un cycle, mais l'ordre de l'analyse change à chaque fois qu'on fait une planification. Il s'agit en fait de 2 boucles imbriquées : la boucle extérieure est sur les sorties et la boucle intérieure sur les entrées. L'ordre d'analyse établit la priorité. La priorité est initialement donnée à la connexion 0-0 (entrée-sortie). À chaque planification, on fait un incrément sous forme de « round-robin » sur le numéro de port de sortie qui obtient la plus haute priorité (pour toutes les entrées). Une fois que tous les ports de sortie ont reçu la priorité la plus élevée, on fait également l'incrément sur l'entrée. Donc, pour un tissu à 4 ports par exemple, après avoir donné les priorités aux connexions 0-0 (entrée-sortie), 0-1, 0-2 et 0-3 (4 planifications différentes), on passe ensuite à la connexion 1-0 qui devient dominante pour la planification suivante et ainsi de suite jusqu'à ce que la priorité soit donnée à la connexion 4-4. Ensuite, à la prochaine planification, la priorité est donnée de nouveau à la connexion 0-0. Cela est fait en analysant tout de même les 16 possibilités lors de chaque planification. Ainsi, lors d'une même planification, l'analyse de chacun des ports de sortie se fait dans un ordre constant pour chacun des ports d'entrée.

Le poids donné à chaque entrée par l'algorithme d'ordonnancement WRR doit être ajusté en conséquence du trafic aux diverses entrées. Un module VOQ (par port d'entrée) ne peut envoyer qu'un seul paquet à la fois. De même, un port de sortie du TI ne peut recevoir qu'un paquet à la fois. Ceci concorde avec l'architecture de type commutateur crossbar.

Lorsqu'une sortie est libérée, le mécanisme de requête d'ordonnancement doit spécifier quelle sortie est libérée (« *outp_freed* »). En effet, cela est nécessaire car toutes les sorties ne sont pas libérées en même temps. Certaines connexions peuvent être en cours de transmission alors que d'autres nécessitent une nouvelle planification. Les connexions en cours ne doivent pas être planifiées à nouveau. De même, l'ordonnanceur

doit spécifier quelle(s) sortie(s) viennent d'être planifiées afin que le module d'envoi de paquets mette à jour sa table d'ordonnancement en conséquence, tout en conservant les tables déjà en transmission.

A.3 Module de gestion des envois de paquet : « send_pkt_ctrl »

Les tableaux a.5 et a.6 décrivent respectivement les paramètres génériques du module « *send_pkt_ctrl* » et ses ports.

Tableau A.5 Description des paramètres génériques du module « send_pkt_ctrl »

Générique	Type	Description
NB_PORT	Entier	Nombre de ports de sortie
NB_PORT_LOG2	Entier	Log_2 de NB_PORT (arrondie vers le haut)
NB_BITS	Entier	Taille du mot d'entrée
NB_BITS_ADDR	Entier	Indique la taille du champ indiquant l'adresse de source/destination incluse dans l'entête du paquet

Tableau A.6 Description des ports du « send_pkt_ctrl »

Champ	Direction	Taille (bits)	Description	Connexions des ports
sys_clk	IN	1	L'horloge	-
sys_reset_n	IN	1	Reset asynchrone actif bas	-
voq_empty	IN	NB_PORT	Indique que le FIFO sélectionné de la VOQ est vide	VOQ, ordonnanceur

Champ	Direction	Taille (bits)	Description	Connexions des ports
voq_length	IN	Table de NB_PORT mots de NB_BITS_ DATA_L bits	Nombre de mots total à retirer du FIFO pour le paquet courant à chacun des ports d'entrée	VOQ, send_pkt_ctrl,
outp_read	IN	NB_PORT	Indique si la sortie est prête à lire un paquet	Entrée du TI, ordonnanceur
scheduled_outp	IN	table d'entiers	Table de décision de planification de ports d'entrée par sortie	Ordonnanceur
scheduled_outp_nb_packet	IN	table d'entiers	Nombre de paquets planifiés pour la connexion courante	Ordonnanceur
scheduled_inp (ne plus utiliser : pas nécessaire comme port)	IN	table d'entiers	Table de décision de planification de port de sortie par entrée	Ordonnanceur
do_schedule_voq_done	IN	1	Indique une nouvelle planification suite à une requête de VOQ	Ordonnanceur
do_schedule_outpfreed_done	IN	1	Indique une nouvelle planification suite à une sortie libérée	Ordonnanceur

Champ	Direction	Taille (bits)	Description	Connexions des ports
do_schedule_new_ outpread_done	IN	1	Indique une nouvelle planification suite à une nouvelle lecture de paquets en sortie du TI	Ordonnanceur
do_schedule_port_done	IN	NB_PORT	Indique quels ports ont étés re-planifié	Ordonnanceur
crossbar_ctrl_r	OUT	NB_PORT mots de NB_PORT_LOG 2 bits	Entrée de contrôle qui indique quelle entrée sélectionner par port de sortie	Commutateur crossbar
crossbar_valid_r	OUT	NB_PORT	Spécification de la validité des sorties	Commutateur crossbar
voq_ctrl	OUT	NB_PORT de NB_PORT_LOG 2 bits	Il précise le mot à lire du FIFO correspondant	VOQ (pop_num)
voq_sel	OUT	NB_PORT	Il indique qu'on veut effectuer une lecture dans un FIFO.	VOQ (pop)
do_schedule_outpfreed	OUT	1	Demande une nouvelle planification suite à une sortie libérée	Ordonnanceur
outp_freed	OUT	NB_PORT	Indique quelles sorties sont libérées	Ordonnanceur
start_of_packet_pop	OUT	Tableau de NB_PORT éléments de NB_PORT bits	Indique le début d'une lecture de paquet du FIFO correspondant	VOQ

Ce module est aussi registré en sortie. Il doit détenir une copie locale des signaux de contrôle qu'il met à jour au fur et à mesure que les mots sont envoyés vers la sortie. Il avise l'ordonnanceur lorsqu'une sortie se libère (« *do_schedule_outpfreed* »). Il lui spécifie laquelle des sorties également (« *outp_freed* »). L'ordonnanceur lui donne ensuite une nouvelle décision de routage en lui spécifiant quel(s) port(s) de sortie a (ont) été(s) programmé(s). Ce module gère donc la transmission des mots d'un certain nombre de paquets selon la planification obtenue précédemment de l'ordonnanceur. Il doit également indiquer au module de VOQ lorsqu'un nouveau paquet est lu pour que celui-ci mette à jour ses tables indiquant le nombre de paquets dans chacun de ses FIFO. Le reste de la communication vers les modules VOQ se fait avec un bit de sélection indiquant quel module VOQ est sélectionné. Chacun des bits du *mot « voq_sel »* correspond à la sélection d'un module VOQ. Ensuite, le signal de contrôle « *voq_ctrl* » indique quel FIFO utiliser à l'intérieur de chacun des modules VOQ sélectionnés. Ce module gère également le commutateur crossbar qui reçoit également ses données des modules VOQ. Le signal de contrôle du commutateur crossbar spécifie quelle entrée utiliser pour chacune des sorties alors que les signaux de validité indiquent si la sortie est active ou non au moment présent.

A.4 Module d'envoi des paquets vers les sorties : commutateur crossbar

Les tableaux a.7 et a.8 décrivent respectivement les paramètres génériques du module commutateur crossbar et ses ports.

Tableau A.7 Description des paramètres génériques du commutateur crossbar

Générique	Type	Description
NB_PORT	Entier	Nombre de ports de sortie
NB_PORT_LOG2	Entier	Log_2 de NB_PORT (arrondie vers le haut)
NB_BITS	Entier	Taille du mot d'entrée

Tableau A.8 Description des ports du commutateur crossbar

Champ	Direction	Taille (bits)	Description	Connexions des ports
in_port	IN	NB_PORT mots de NB_BITS	Données d'entrée provenant des VOQ	VOQ
in_valid	IN	NB_PORT	Spécification de la validité des sorties	send_pkt_ctrl,
in_ctrl	IN	NB_PORT mots de NB_PORT_LOG2 bits	Entrée de contrôle qui indique quelle entrée sélectionner par port de sortie	send_pkt_ctrl,
out_port	OUT	NB_PORT mots de NB_BITS	Données de sortie du TI	La sortie du TI
out_valid	OUT	NB_PORT	Validité des sorties	La sortie du TI

Ce module fait simplement router les sorties selon les commandes reçues du module de gestion d'envoi de paquets. Ce module est combinatoire. Une entrée peut transmettre vers une seule sortie à la fois. De même, une sortie ne peut recevoir qu'un paquet à la fois. Il indique également la validité de la sortie pour chacun des ports.

Les divers sous-modules du TI matériel sont détaillés dans la présente annexe. L'annexe suivant présente un exemple du format des données obtenues après une simulation avec l'environnement de validation à haut niveau.

ANNEXE B. FORMAT DES DONNÉES DE SIMULATION

Cette section a pour but de présenter un exemple des résultats de simulation obtenus après une simulation avec l'environnement à haut niveau présenté au chapitre 2. La première partie du fichier montre les paramètres de configuration propres à la simulation tels que présenté au chapitre 2. Il s'agit de la plupart des éléments présentés aux tableaux 2.1, 2.2 et 2.3. Cela inclut la configuration de l'architecture, du modèle de paquet et du trafic généré. Ensuite, on observe les diverses lectures, écritures et pertes de paquets accompagnées des informations contenues dans le paquet et du temps de simulation auquel cet événement arrive. À chaque cycle de simulation, l'état des divers FIFO est également imprimé. Dans l'exemple donné, il s'agit des lignes identifiées avec la mention *« FIFO_sate iport »* et de la ligne identifiée avec la mention *« OQ_state »*. Ces données représentent l'état des VOQ par port d'entrée et l'état des FIFO de sortie respectivement. Dans le cas des VOQ, pour un même temps de simulation, chaque ligne montre l'état des VOQ pour une entrée. Pour chaque ligne montrant l'état des FIFO, le nombre de paquets en mémoire par sortie est présenté par ordre croissant de port de sortie.

...

*# **
SF_Arch_Cfg: WRR NO_PATH_SHARING 4 ports 1.2 SU HIGH_LEVEL 1 1 1
1 weight_RR 100 fifo_depth 56 bus_width 4 ns_clk
Packet_Model_Cfg: 8 addr_w 14 pack_t_w 8 payl_w 10 data_l_w 48 full_
header_w
Traffic_Cfg: RANDOM_DEST 50 bw_min_dl 90 bw_max_dl 100 bw_perc_bw 0
init_cycles 1 1 1 1 mul_factor_in_bw 1 1 1 1 mul_factor_in_dl
**
...
...*

*# at time 18236 ns: receive packet : { src_addr: 3 dest_addr: 0 packet_id: 384 payload: 0 data_length: 52
data_buffer {
641770E7BE6E21D1664E38C755CE3143CA1AFC1106A08C72CF352BAD00CE544AD897F835A70F1A81
24EAD392492EFEF225161D9D }
# }
# at time 18236 ns: send packet : { src_addr: 1 dest_addr: 2 packet_id: 425 payload: 0 data_length
: 63 data_buffer {
1B223D6B277EA5B75F8656D13D678BB9AE859333B4A551B9D281CCB57377000894DAACB646405207DDB
21B88F35F10F86579D26C45E6122BF670C9B2146622 }
# }
# FIFO_state iport- 0 : 35 8 33 6 at time : 18238 ns
# FIFO_state iport- 1 : 1 29 18 25 at time : 18238 ns
# FIFO_state iport- 2 : 8 30 7 28 at time : 18238 ns
# FIFO_state iport- 3 : 10 10 13 21 at time : 18238 ns
# OQ_FIFO_state 5 5 5 5
# at time 18240 ns: receive packet : { src_addr: 2 dest_addr: 2 packet_id: 406 payload: 0 data_len
gth: 55 data_buffer {
B2B4C2F065963893F6FC7D30005FAF2F55AD31E1F7E6D82910715FEE6B05F45B986018FFBCA4638E
0E15CFFD2C7C788AD477C2B11B5663 }*

...
...

*# Packet drop iport : 1 at :109168ns
Packet drop dataiport : { src_addr: 1 dest_addr: 0 packet_id: 3131 payload: 0 data_length: 37 data_buffer {
A170B4341F32C2C9BC41FF4F14276BE216B08810980FFB9D1196AF60EDD1C1B40DBCB50854 }*

...
...